



SUPRA SERVER PDM

Programming Guide
(UNIX & VMS)

P25-0240-49




SUPRA® Server PDM Programming Guide (UNIX & VMS)

Publication Number P25-0240-49

© 1990–1992, 1994–2002 Cincom Systems, Inc.
All rights reserved

This document contains unpublished, confidential, and proprietary information of Cincom. No disclosure or use of any portion of the contents of these materials may be made without the express written consent of Cincom.

The following are trademarks, registered trademarks, or service marks of Cincom Systems, Inc.:

AD/Advantage®	iD CinDoc™	MANTIS®
C+A-RE™	iD CinDoc Web™	Socrates®
CINCOM®	iD Consulting™	Socrates® XML
Cincom Encompass®	iD Correspondence™	SPECTRA™
Cincom Smalltalk™	iD Correspondence Express™	SUPRA®
Cincom SupportWeb®	iD Environment™	SUPRA® Server
CINCOM SYSTEMS®	iD Solutions™	Visual Smalltalk®
 gOOi™	intelligent Document Solutions™	VisualWorks®
	Intermax™	

UniSQL™ is a trademark of UniSQL, Inc.
ObjectStudio® is a registered trademark of CinMark Systems, Inc.

All other trademarks are trademarks or registered trademarks of their respective companies.

Cincom Systems, Inc.
55 Merchant Street
Cincinnati, Ohio 45246-3732
U.S.A.

PHONE: (513) 612-2300
FAX: (513) 612-2000
WORLD WIDE WEB: <http://www.cincom.com>

Attention:

Some Cincom products, programs, or services referred to in this publication may not be available in all countries in which Cincom does business. Additionally, some Cincom products, programs, or services may not be available for all operating systems or all product releases. Contact your Cincom representative to be certain the items are available to you.

Release information for this manual

The *SUPRA Server PDM Programming Guide (UNIX & VMS)*, P25-0240-49, is dated January 15, 2002. This document supports Release 2.4 of SUPRA Server under OpenVMS/Alpha and VMS/VAX (with HDMP support) and Release 1.3 under UNIX with PDM support.

We welcome your comments

We encourage critiques concerning the technical content and organization of this manual. Please take the [survey](#) provided with the online documentation at your convenience.

Cincom Technical Support for SUPRA Server PDM

FAX: (513) 612-2000
Attn: SUPRA Server Support

E-mail: helpna@cincom.com

Phone: 1-800-727-3525

Mail: Cincom Systems, Inc.
Attn: SUPRA Server Support
55 Merchant Street
Cincinnati, OH 45246-3732
U.S.A.

Contents

About this book	xi
Using this document.....	xi
Document organization	xii
Conventions	xiii
SUPRA Server documentation series	xvi
 Understanding SUPRA Server from a programming viewpoint	 19
Overview	19
The Relational Data Manager (RDM) (VMS)	22
The Directory.....	23
The Physical Data Manager (PDM).....	23
MANTIS.....	24
SPECTRA (VMS/VAX).....	24
 Understanding the Relational Data Manager and Relational Data Manipulation Language (VMS)	 25
Understanding views	27
Understanding how your DBA creates and maintains views.....	29
Defining views	29
Changing view design	29
Changing database structure	29
Understanding Relational Data Manipulation Language	30
Signing on/off	30
Manipulating data	30
Controlling data recovery	33
Using special function RDML statements	33
Testing views using the DBAID Utility subset.....	34

Using the DBAID Utility subset to test views (VMS)	35
Accessing DBAID	36
Signing on to DBAID	40
Accessing the SUPRA Server HELP facility	41
* command	42
= command	42
BYE command	43
BY-LEVEL command	43
CAUTIOUS command	45
COLUMN-DEFN command	46
COLUMN-TEXT command	51
COMMIT command	52
DELETE command	53
ERASE command	56
FIELD-DEFN command	56
FIELD-TEXT command	59
FORGET command	61
GET command	62
GO command	68
INSERT command	72
KEEP command	77
LINESIZE command	78
MARK command	79
MARKS command	80
OPEN command	81
PAGESIZE command	83
PRINT-STATS command	84
RELEASE command	85
RESET command	86
SHOW-NAVIGATION command	87
SIGN-OFF command	88
SIGN-ON command	89
STATS command	90
STATS-OFF command	91
STATS-ON command	92
SURE command	93
UNDEFINE command	94
UPDATE command	95
USER-LIST command	98
VIEW-DEFN command	99
VIEWS command	101
VIEWS-FOR-USER command	102

Writing an RDM program in COBOL, FORTRAN, and BASIC (VMS) 103

Understanding RDML statement format	103
Enrolling your program in the SUPRA directory	108
Writing the identification division (COBOL)	108
Writing the program naming statement (FORTRAN and BASIC)	108
Defining program data	109
Writing the environment division (COBOL)	109
Writing the Data Division (COBOL) and the Declaration Statements (FORTRAN and BASIC)	110
Checking for current program	121
Defining program logic	122
Signing on/off	122
Retrieving rows	123
Modifying rows	134
Controlling database recovery	140
Handling error conditions	141
Implementing and executing an RDM program	149

Coding RDM program statements (VMS) 153

Coding program data statements	153
INCLUDE <i>view-data</i>	153
INCLUDE ULT-CONTROL	163
Coding program logic statements	167
COMMIT	167
DELETE	169
FORGET	173
GET	176
INSERT	190
MARK	195
RELEASE	198
RESET	200
SIGN-OFF	202
SIGN-ON	205
UPDATE	210

Understanding Physical Data Manipulation Language (PDML)	213
Opening/closing data sets	214
Adding a primary record	215
Reading a primary record	215
Updating a primary record	218
Deleting a primary record	218
Adding a related record	219
Reading a related record	223
Updating a related record	225
Deleting a related record	226
 Using PDML	 227
Table of PDML commands	229
Data list parameter keywords	233
PDML commands	238
ADD-M	238
ADDVA	242
ADDVB	249
ADDVC	256
ADDVR	263
CNTRL (VMS only)	269
COMIT	273
DEL-M	276
DELVD	278
MARKL	283
OPCOM	285
RDNXT	289
READD	294
READM	300
READR	303
READV	309
READX	314
RESET	324
RQLOC	326
SINOF	328
SINON	332
WRITM	338
WRITV	342

Optimizing your PDML program	347
Linking your application program	348
Using logical units of work	350
Reserving resources	350
Implementing a logical unit of work	351
Understanding deadlocks and how to prevent them	352
Handling errors in a logical unit of work	353
Managing your application program	354
Communicating with SUPRA Server	354
Parameter list definitions	356
Initialization and termination requirements	358
Task management	358
Checking the <i>status</i> parameter	359
Using standard primary data-set processing	360
Data items you must not refer to	360
The ADD-M command	361
Structural maintenance during serial processing	362
Using standard related-data-set processing	362
The reference parameter	363
Improving program efficiency	366
Understanding RDNXT serial processing	367
Testing database programs	368
Using extended data item processing	369
Data-item binding	369
Code-directed reading	370
PDML examples	371
 Sample RDM programs (VMS)	 375
Sample COBOL RDM program	375
Sample FORTRAN RDM program	391
Sample BASIC RDM program	407
 Index	 423

About this book

Using this document

The information in this guide is directed toward application programmers responsible for manipulating data held on SUPRA Server databases.

The *SUPRA Server PDM Programming Guide (UNIX & VMS)* provides you with the commands you need to access and manipulate the database for your application programs.

If you operate under VMS, this manual helps you:

- ◆ Understand the Relational Data Manager (RDM)
- ◆ Write an RDM program in COBOL, FORTRAN, and BASIC
- ◆ Understand the Physical Data Manager (PDM)
- ◆ Use Physical Data Manipulation Language (PDML)
- ◆ Use the DBAID utility subset

If you operate under UNIX, this manual helps you:

- ◆ Understand the Physical Data Manager (PDM)
- ◆ Use Physical Data Manipulation Language (PDML)

Document organization

The information in this guide is organized as follows:

Chapter 1—Understanding SUPRA Server from a programming viewpoint

Describes the Relational Data Manager (RDM), the Directory, the Physical Data Manager (PDM), MANTIS, and SPECTRA.

Chapter 2—Understanding the Relational Data Manager and Relational Data Manipulation Language (VMS)

Describes views, how to create, maintain, and test them, and Relational Data Manipulation Language.

Chapter 3—Using the DBAID utility subset to test views (VMS)

Describes DBAID utility commands to use for testing a view before testing with applications.

Chapter 4—Writing an RDM program in COBOL, FORTRAN, and BASIC (VMS)

Describes how to write RDM programs in COBOL, FORTRAN and BASIC.

Chapter 5—Coding RDM program statements (VMS)

Provides a listing of RDM statements in two groups: program data statements and program logic statements.

Chapter 6—Understanding Physical Data Manipulation Language (PDML)

Describes how to use PDML to access and manipulate a database from an application program.

Chapter 7—Using PDML

Describes how to use PDML to access the PDM and manipulate data on your SUPRA Server physical databases.

Chapter 8—Optimizing your PDML program

Summarizes common programming procedures that will help you make the best use of your PDML program.

Appendix A—Sample RDM programs (VMS)

Provides sample RDM programs demonstrating the use of RDM statements for COBOL, FORTRAN and BASIC.

Index

Conventions

The following table describes the conventions used in this document series:

Convention	Description	Example
Constant width type	Represents screen images and segments of code.	<pre>PUT 'customer.dat' GET 'miller\customer.dat' PUT '\DEV\RMT0'</pre>
Slashed b (<i>b</i>)	Indicates a space (blank). The example indicates that four spaces appear between the keywords.	<pre>BEGINxxxxSERIAL</pre>
Brackets []	Indicate optional selection of parameters. (Do not attempt to enter brackets or to stack parameters.) Brackets indicate one of the following situations:	
	A single item enclosed by brackets indicates that the item is optional and can be omitted. The example indicates that you can optionally enter a WHERE clause.	<pre>[WHERE <i>search-condition</i>]</pre>
	Stacked items enclosed by brackets represent optional alternatives, one of which can be selected. The example indicates that you can optionally enter either WAIT or NOWAIT. (WAIT is underlined to signify that it is the default.)	<pre>[<u>(WAIT)</u> (NOWAIT)]</pre>

Convention	Description	Example
Braces { }	<p>Indicate selection of parameters. (Do not attempt to enter braces or to stack parameters.) Braces surrounding stacked items represent alternatives, one of which you must select.</p> <p>The example indicates that you must enter ON or OFF when using the MONITOR statement.</p>	<code>MONITOR {ON } {OFF }</code>
<u>Underlining</u> (In syntax)	<p>Indicates the default value supplied when you omit a parameter.</p> <p>The example indicates that if you do not choose a parameter, the system defaults to WAIT.</p>	<code>[(WAIT) (NOWAIT)]</code>
	<p>Underlining also indicates an allowable abbreviation or the shortest truncation allowed.</p> <p>The example indicates that you can enter either STAT or STATISTICS.</p>	<code><u>STATISTICS</u></code>
Ellipsis points...	<p>Indicate that the preceding item can be repeated.</p> <p>The example indicates that you can enter multiple host variables and associated indicator variables.</p>	<code>INTO :host-variable [:ind- variable],...</code>

Convention	Description	Example
UPPERCASE lowercase	In most operating environments, keywords are not case-sensitive, and they are represented in uppercase. You can enter them in either uppercase or lowercase.	<code>COPY MY_DATA.SEQ</code> <code>HOLD_DATA.SEQ</code>
	In the UNIX operating environment, keywords are case-sensitive, and you must enter them exactly as shown.	<code>cp *.QAR /backup</code>
<i>Italics</i>	Indicate variables you replace with a value, a column name, a file name, and so on. The example indicates that you must substitute the name of a table.	<code>FROM table-name</code>
Punctuation marks	Indicate required syntax that you must code exactly as presented. () parentheses . period , comma : colon ' ' single quotation marks	<code>(user-id, password, db-name)</code> <code>INFILE 'Cust.Memo' CONTROL</code> <code>LEN4</code>
SMALL CAPS	Represent a keystroke. Multiple keystrokes are hyphenated.	ALT-TAB
UNIX VMS	Information specific to a certain operating system is flagged by a symbol in a shadowed box (UNIX) indicating which operating system is being discussed. Skip any information that does not pertain to your environment.	UNIX To delete these files, return to the shell and use the <code>rm</code> command. VMS To delete these files, return to the command level and use the <code>DELETE</code> command.

SUPRA Server documentation series

SUPRA Server is the advanced relational database management system for high-volume, update-oriented production processing. A number of tools are available with SUPRA Server including DBA Functions, DBAID, precompilers, SPECTRA, and MANTIS. The following list shows the manuals and tools used to fulfill the data management and retrieval requirements for various tasks. Some of these tools are optional. Therefore, you may not have all the manuals listed. For a brief synopsis of each manual, refer to the *SUPRA Server PDM Digest for VMS Systems*, P25-9062.

Overview

- ◆ *SUPRA Server PDM Digest for VMS Systems*, P25-9062

Getting started

- ◆ *SUPRA Server PDM UNIX Installation Guide*, P25-1008
- ◆ *SUPRA Server PDM VMS Installation Guide*, P25-0147
- ◆ *SUPRA Server PDM UNIX Tutorial*, T25-2262
- ◆ *SUPRA Server PDM VMS Tutorial*, T25-2263

General use

- ◆ *SUPRA Server PDM Glossary*, P26-0675
- ◆ *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022

Database administration tasks

- ◆ *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260
- ◆ *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130
- ◆ *SUPRA Server PDM System Administration Guide (UNIX)*, P25-0132*
- ◆ *SUPRA Server PDM Utilities Reference Manual (UNIX & VMS)*, P25-6220

- ◆ *SUPRA Server PDM Directory Views (VMS)*, P25-1120
- ◆ *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500*
- ◆ *SPECTRA Administrator's Guide*, P26-9220**

Application programming tasks

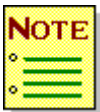
- ◆ *SUPRA Server PDM Programming Guide (UNIX & VMS)*, P25-0240
- ◆ *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130
- ◆ *SUPRA Server PDM System Administration Guide (UNIX)*, P25-0132*
- ◆ *SUPRA Server PDM RDM Administration Guide (VMS)*, P25-8220
- ◆ *SUPRA Server PDM Windows Client Support User's Guide*, P26-7500*
- ◆ *MANTIS Planning Guide*, P25-1315**

Report tasks

- ◆ *SPECTRA User's Guide*, P26-9561**



Manuals marked with an asterisk (*) are listed twice because you use them for different tasks.



Educational material is available from your regional Cincom education department.

1

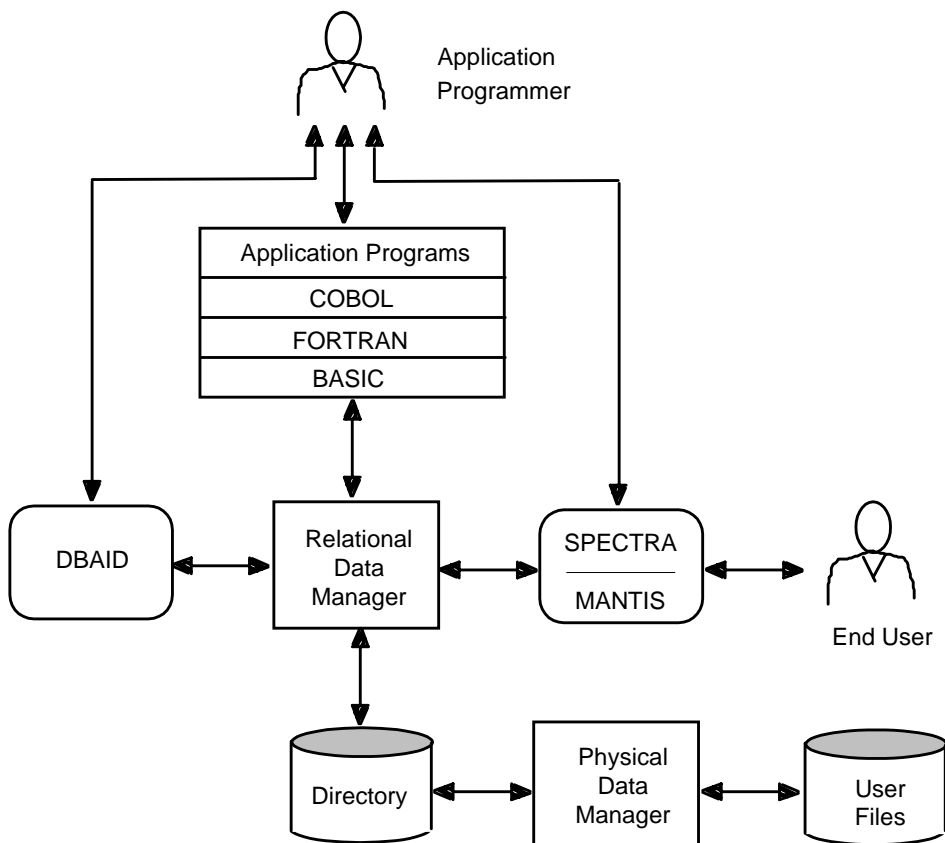
Understanding SUPRA Server from a programming viewpoint

Overview

SUPRA Server is an interactive database system that allows you to use advanced features for control of data resources and high programming productivity. You can use SUPRA Server in VMS and UNIX environments.

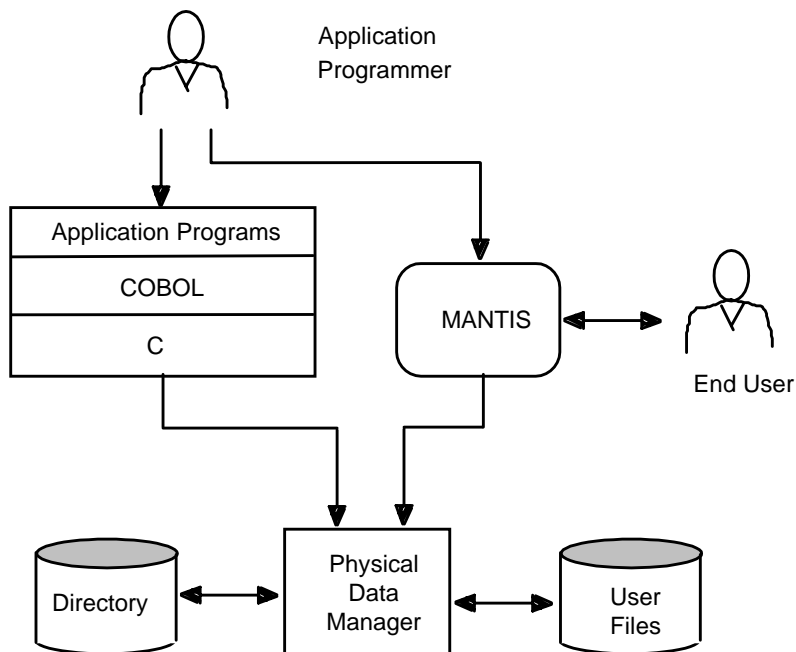
VMS

SUPRA Server provides integrated control through the following tools: Relational Data Manager (RDM), Directory, Physical Data Manager (PDM), MANTIS, and SPECTRA (VMS/VAX). The following figure illustrates the programmer's view of SUPRA Server in VMS environments.



UNIX

SUPRA Server provides integrated control through the following tools: Directory, Physical Data Manager (PDM), and MANTIS. The following figure illustrates the programmer's view of SUPRA Server components in UNIX environments.



The Relational Data Manager (RDM) (VMS)

SUPRA Server provides a high-level Relational Data Manipulation Language (RDML) consisting of four basic commands: GET, UPDATE, INSERT, and DELETE. The Relational Data Manager (RDM) carries out these and other operations necessary to retrieve and manipulate the columns requested by your program. You can use these commands to read and modify data held on PDM and RMS data sets.

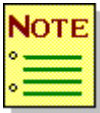
After carrying out retrieval and maintenance operations, the RDM presents the columns to your application program in the required format. RDM checks data integrity and notifies the program if recovery is required. Because application programs deal only with the relational view of data, the database is completely insulated from change in physical data structure, data storage media, or physical data-set relationships.

In SUPRA Server, all data looks like a relational, two-dimensional table. You access data at the logical level using defined views or derived relations stored on the Directory as views. A view is a group of logical records or rows making up a table of data.

The RDM preprocessor validates the views accessed by your program and generates appropriate data areas; and then it converts the high-level logic statements into calls to the RDM run-time system. These calls perform the actions requested, accessing the database as defined by the view definitions stored on the Directory.

The Directory

The Directory integrates and controls SUPRA Server components. The RDM manages the logical structure of data, and the PDM manages the physical structure. The Directory holds a description of both structures and how they map to each other. It contains metadata (data about data) rather than your business data. Metadata includes definitions of all the databases, data sets, views, users, and so on, available within SUPRA Server. Your DBA maintains these definitions using Directory maintenance or DBA functions.



Directory views are not available in UNIX environments.

The Physical Data Manager (PDM)

The Physical Data Manager (PDM) manages access to data in databases. Your DBA defines the database on the Directory. For efficient use by the PDM, DBA consolidates this definition into a table called the database description.

The PDM also supports the following recovery techniques:

- ◆ **Task Level Recovery (TLR).** Available to all programs and built into the PDM. TLR allows processing to be split into logical units of work that can be guaranteed to be fully completed or fully undone. This feature maintains database integrity.
- ◆ **System Logging.** Allows recovery from a device failure. This type of recovery uses a log file, containing all update and control functions, and a task log, containing before-images. TLR must be used in conjunction with system logging.
- ◆ **Shadow Recording.** Also allows recovery from device failures and is an alternative to system logging. Shadow recording can maintain two identical copies of data sets in the database at the same time. Each time you update a data set in the main database, an identical record is written to the second (shadow) data set.

MANTIS

MANTIS is an application development system that consists of design facilities (screen and file) and a programming language. You use MANTIS to design and create formatted screens and permanent files for storing and manipulating data.

SPECTRA (VMS/VAX)

SPECTRA is a language that allows you to easily retrieve and update information held on a SUPRA Server database. Using SPECTRA, you can produce reports and update files by adding, changing, and deleting information. You write processes in SPECTRA using views defined on the Directory and therefore, giving you all the advantages of the RDM.

The SPECTRA filing system stores information about your organization in central and personal files. Central files contain all the data for your company or organization. They are database views and are shared by all users. Because central files often contain sensitive information, your access to them may be limited depending on your responsibilities. Personal files are files you create for your specific needs using information from available central files or data from other sources. Your personal files are not available to other SPECTRA users unless you share them.

SPECTRA's personal file system also contains online documentation and any processes that have been saved. All SPECTRA commands apply equally to central and personal files.

A SPECTRA process may combine central and personal files as well as external RMS files as required. SPECTRA accesses external files sequentially.

2

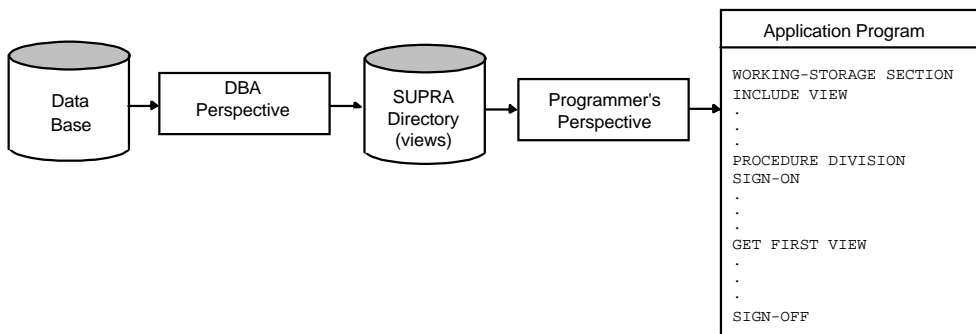
Understanding the Relational Data Manager and Relational Data Manipulation Language (VMS)

The Relational Data Manager (RDM) allows you to write an application program without knowing the physical structure of the database. RDM retrieves the data you need while providing database security and integrity. It does this by logically retrieving the data and presenting it in a view.

A view is a table containing logical data items that the RDM derives from different physical locations. Your DBA designs these views and builds in constraints to prevent you from accidentally destroying the integrity of the database.

Before accessing views, RDM requires that your DBA first determine what the database looks like, how it can be accessed, what values go into it, and who can access it.

You use COBOL, FORTRAN, or BASIC programs which use Relational Data Manipulation Language (RDML) maintenance commands to access RDM and manipulate the database. The following figure illustrates the programmer's view of the database when using RDM:



The following list shows the sequence of events between the time you write the program and when RDM presents the data you requested:

1. You write the program using views your DBA assigns you.
2. RDML preprocessors use the SUPRA Directory database, SUPRAD, to supply working storage definitions and to convert the high-level RDML into standard source statements.
3. Standard compilers then convert it into object code.
4. You link the object code into an executable or shareable image.
5. When your program executes, the Directory uses the physical data descriptions, and RDM uses the logical data descriptions to access the database and present the data in the view requested by the application program.
6. A source language preprocessor updates the SUPRAD database with the following information:
 - When your program was preprocessed
 - What language your program is written in
 - Which views your program uses

This updating is called program enrollment. In addition to performing program enrollment, the preprocessors also obtain definitions of views from the Directory.

To use the RDM, you need to fully understand views, how your DBA creates and maintains views, and the RDML.

Understanding views

A view is a table containing logical data that the RDM derives from physical locations on the database using the RDML commands in your COBOL, BASIC, or FORTRAN program. A view contains rows and columns, as shown in the following figure:

CUSTOMER-ORDER-VIEW							
	Order Number	Customer Number	Part Number	Quantity Ordered	Part Cost	Total Cost	Ship Date
	1750	175610	02753	500	50.00	25000.00	02/10/95
ROW	1751	185910	02984	40	10.00	400.00	03/12/95
	1752	176231	17642	100	20.00	2000.00	03/15/95

↑
COLUMN

Although a view resembles a flat file, there are two important differences:

- ◆ The ordering of rows within the file is not always controlled by your maintenance operations
- ◆ Columns can have null values

Once your DBA has defined the columns included in a view, you can use all of the view or a subset of the view (a user view). A subset of a view uses only some of the columns in a view and/or reorders a view to meet specific needs. User views should be used whenever possible to improve performance and to reduce resource requirements. The following table illustrates a view and a user view based on the information in the preceding figure:

CUSTOMER-ORDER-VIEW (Entire view)	USER-VIEW (Subset)
Order Number	
Customer Number	
Part Number	Part Number
Quantity Ordered	Quantity Ordered
Part Cost	Part Cost
Total Cost	
Ship Date	

Understanding how your DBA creates and maintains views

The DBA's responsibility to the database is to:

- ◆ Describe the logical and physical attributes of data
- ◆ Define the relationships that exist between units of data
- ◆ Allow for physical access to the data
- ◆ Provide security/integrity constraints for the database
- ◆ Optimize system performance

One of the ways the DBA accomplishes these tasks is by designing and maintaining views.

Defining views

Your DBA defines the data that should be in each view and how that data (rows) should be accessed. Your DBA determines which columns contain fixed values and defines unique and nonunique keys. (For more information on unique and nonunique keys and required columns, see [“Manipulating data”](#) on page 30.) Because using RDML commands to modify data in a view also modifies user data in the database, your DBA also decides the access you have to the data in the row (read-only, update, etc.).

Changing view design

At any time, your DBA can alter existing views, add new views, change keys or required columns, and add information to or delete information from existing views. The DBA has control over changes made to your view definition and should distribute copies of definitions and changes that affect you.

Changing database structure

Your DBA can make many changes to the structure of the database or add new data items, without requiring you to change or re-compile programs. However, if your DBA changes the length of data items or deletes a data item(s), you must re-compile programs and rebuild images that use them if these programs use views containing these data items. In this case, other changes might be required as well.

Understanding Relational Data Manipulation Language

The RDML is a high-level language you use to sign on and off the system, to manipulate data, and to control data recovery. There are also a few special function statements. Before you run an RDML application program, first link it with the RDM run-time system as described in “[Implementing and executing an RDM program](#)” on page 149. For detailed information on syntax and usage considerations, see “[Coding RDM program statements \(VMS\)](#)” on page 153.

Signing on/off

Use SIGN-ON to establish communication between your program and the RDM. It identifies you as the user and allows access to views based on your user name. Before issuing a SIGN-ON, have your program call the routine CSV_SETUP_REALM. This routine allows you to specify the open mode only for those data sets required by your application program. By calling this routine, you avoid possibly impairing system performance since you are not opening files that your program does not require.

SIGN-OFF informs the RDM that you want to terminate your session.

Manipulating data

You use these RDM statements to perform maintenance on a view and manipulate all data on an occurrence-by-occurrence basis:

- ◆ **DELETE.** Removes a row from the view (see “[Using DELETE](#)” on page 136 and “[DELETE](#)” on page 169).
- ◆ **GET.** Retrieves a row from the view (see “[Using GET to control record holding](#)” on page 132 and “[GET](#)” on page 176).
- ◆ **INSERT.** Inserts a new row into the view (see “[Using INSERT](#)” on page 138 and “[INSERT](#)” on page 190).
- ◆ **UPDATE.** Updates column values in an existing row in the view (see “[Using UPDATE](#)” on page 135 and “[UPDATE](#)” on page 210).

Before performing maintenance on a view, you must first retrieve it. You retrieve rows using designated key values.

Retrieving rows sequentially using the GET statement

You can use GET to retrieve rows by the order in which they appear in the physical files. For detailed information on using GET, see “GET” on page 176.

Retrieving rows using keys defined by your DBA

Each view contains one or more columns that your DBA may designate as unique or nonunique keys to the view. You can access a set of rows by assigning values to the keys of the view (if any exist). You use the keys to locate a specific row or to perform a generic read. Both types of reads return all qualifying rows from the view, one at a time.

Your DBA can also assign fixed values, or constants, to impose constraints on the program and thus limit your application to retrieve or update selected rows. In addition, required columns must be present when RDM is constructing the row; otherwise, the row is skipped and not retrieved.

Unique keys

A *simple unique key* is a single column designated as the key to that view. Using this key value, you can select and retrieve data. A key must have a valid, non-null value.

A *compound unique key* is several columns designated as unique logical keys, and the combination of the key values is unique. This implies a connection between the columns. For example, to check customer orders for a certain part number, you use a view with both customer number and part number as key values. RDM retrieves the specific customer number and part number combination if it is present.

Nonunique keys

A *nonunique key* allows more than one row to contain the same key. An example is a customer file with a list of notes or comments concerning each customer. To retrieve a list of comments for a customer, you can define the customer number as a single nonunique key. When the program does its first GET using a customer number, it retrieves the first comment for that customer. A subsequent GET retrieves the second comment; the third GET the third comment; and so on.

A *compound nonunique key* is more than one column designated as a nonunique key. However, all of the nonunique keys together still do not completely describe the row occurrence as unique. You can still have more than one row with the same compound nonunique key.

If your DBA does not specify a column as a nonunique key, you can only retrieve the rows sequentially, not based on a value. Your DBA can define nonunique keys in two ways:

- ◆ Omit a key that would uniquely define the view.
- ◆ Explicitly define a column as a nonunique key. Then you can perform searches based on that value.

Assigning values to unique and nonunique keys

Your DBA can define a logical key with a fixed value by using the keyword `CONST` in the column definition and then assigning a literal value to the column. You cannot change the value of such columns. RDM uses this value as if the program had supplied it as a key. RDM treats a `CONST` key as a `NONUNIQUE` key with the fixed value supplied unless you specify the keyword `UNIQUE` before `CONST`.

For example, if your DBA defined a `CONST` value of `TN` in the state column, then, no matter what you do, the program can retrieve and update only Tennessee customers.

Required columns

A required column is one which must contain a valid and non-null value for the row to be included in the view. A column is not required by default and does not need a value. If you want to indicate null, insert the `NASI` (see “[Validating data](#)” on page 115). Blanks signify a null value for zoned and packed data items. All logical keys are required columns—they must have a valid non-null value.

In summary, your DBA defines certain columns as required, keys, or fixed values. Remember, columns defined as keys or fixed values are also considered required columns.

Controlling data recovery

COMMIT and RESET control task-level database recovery. The statements function differently, depending on the environment and recovery system supported. For more information on the use of RDML, see “[Writing an RDM program in COBOL, FORTRAN, and BASIC \(VMS\)](#)” on page 103.

Using special function RDML statements

You can use the following special RDML statements :

- ◆ **MARK.** Records the current position of the view established by the most recent GET, INSERT, or UPDATE statement (see “[MARK](#)” on page 195).
- ◆ **FORGET.** Removes the specified mark from the list of marks in use (see “[FORGET](#)” on page 173).
- ◆ **RELEASE.** Releases the internal storage space used by a view (see “[RELEASE](#)” on page 198).

Testing views using the DBAID Utility subset

The DBAID Utility subset is an online tool that allows the DBA to define a new view, open the view, issue RDML statements, and examine the results. The DBA can then change the view, if necessary, reorder for efficiency, or try different access methods. These activities have no impact on the Directory database, SUPRAD, until the DBA saves any changes.

Certain DBAID commands are also available to non-DBA users for use when constructing programs that use RDM views. Using the DBAID Utility subset of commands, you can test a view before testing with applications until you are sure that the view is correctly defined. You can also use the DBAID Utility subset as an educational tool for immediate hands-on experience with the views being accessed.

3

Using the DBAID Utility subset to test views (VMS)

A subset of the DBAID Utility commands is available to the application programmer to use for testing a view before testing with applications. Testing views helps you determine how to design applications using views. Using DBAID, you can run test cases until you are satisfied that the view is correctly defined.

DBAID has five types of commands:

- ◆ **RDML commands.** Enable you to use test data with a defined view to make sure the view is properly defined.
- ◆ **Editing commands.** Enable you to define and modify a view.
- ◆ **System commands.** Enable you to display information about the currently executing DBAID Test Facility.
- ◆ **Built-in view commands.** Enable you to inspect the view or obtain information after it is opened.
- ◆ **Statistics commands.** Enable you to start, stop, and display statistics on both the logical and physical levels.

Accessing DBAID

To access the DBAID test facility, you can either: (1) select the DBAID Test Facility from the SUPRA Server facilities screen; or (2) set up a symbol or command file. A command file called RUNDBAID, which is located in the directory SUPRA_COMS, is supplied to invoke DBAID. Use the SUPRA_COMS:SUPRA_SYMBOL.COM procedure to define the RUNDBAID symbol. For example, to invoke DBAID with database TESTDB, you could enter \$RUNDBAID TESTDB or just \$RUNDBAID with no parameters. If you do not enter any parameters and you are not using the logical CSI_SCHEMA, you are prompted for the database name.

If you have an access authority of Database Administrator or Privileged and want to list, edit, save, and so on, a view, then list the view before you open it. This makes the text of the view known to DBAID. Such a view is called a virtual view. You can list any view in this way. However, if you open a view first, you must be authorized to use that view.

The following tables alphabetically list all the commands within each category and provide a brief description and a section reference for detailed information.

RDML commands

Command	Description	Section
=	Reissues the previous RDML command.	"= command" on page 42
BYE	Exits DBAID.	"BYE command" on page 43
CAUTIOUS	Prohibits an automatic COMMIT.	"CAUTIOUS command" on page 45
COMMIT	Issues an RDML COMMIT.	"COMMIT command" on page 52
DELETE	Issues an RDML DELETE.	"DELETE command" on page 53
ERASE	Issues an RDM RESET and returns an X FSI.	"ERASE command" on page 56

Command	Description	Section
FORGET	Removes the specific mark from the list of marks in use.	“FORGET command” on page 61
GET	Issues an RDML GET command that retrieves and displays the requested row.	“GET command” on page 62
GO	Issues multiple RDML GET commands and displays the records in a tabular format.	“GO command” on page 68
INSERT	Issues an RDML INSERT.	“INSERT command” on page 72
KEEP	Prohibits an automatic RESET.	“KEEP command” on page 77
MARK	Issues an RDML MARK. Marks the current position of the <i>view-name</i> established by the previous GET.	“MARK command” on page 79
OPEN	Readies either a virtual or stored view for use.	“OPEN command” on page 81
RELEASE	Issues an RDML RELEASE. Closes all opened views and releases the storage occupied by the views.	“RELEASE command” on page 85
RESET	Issues an RDML RESET.	“RESET command” on page 86
SIGN-OFF	Signs off the user from DBAID.	“SIGN-OFF command” on page 88
SIGN-ON	Identifies the user to DBAID.	“SIGN-ON command” on page 89
SURE	Causes a COMMIT after each successful INSERT, UPDATE or DELETE.	“SURE command” on page 93
UPDATE	Issues an RDML UPDATE.	“UPDATE command” on page 95

Editing commands

Command	Description	Section
UNDEFINE	Removes a defined virtual view.	"UNDEFINE command" on page 94

System commands

Command	Description	Section
*	Used with other commands to indicate the most recent view name used.	"* command" on page 42
LINESIZE	Specifies width of line for DBAID output.	"LINESIZE command" on page 78
MARKS	Lists all the open MARKs and the views they are marking.	"MARKS command" on page 80
PAGESIZE	Specifies the number of lines on the page/screen for DBAID output.	"PAGESIZE command" on page 83
USER-LIST	Displays the attribute list for the view named.	"USER-LIST command" on page 98
VIEWS	Displays all views active in DBAID.	"VIEWS command" on page 101

Built-in view commands

Command	Description	Section
BY-LEVEL	Displays the column names in the view by level of occurrence.	“BY-LEVEL command” on page 43
COLUMN-DEFN	Displays the full description of a column in a view.	“COLUMN-DEFN command” on page 46
COLUMN-TEXT	Displays the short and long text for a column in a view.	“COLUMN-TEXT command” on page 51
FIELD-DEFN	Displays the full description of a column in a view.	“FIELD-DEFN command” on page 56
FIELD-TEXT	Displays the short and long text for a column in a view.	“FIELD-TEXT command” on page 59
VIEW-DEFN	Displays a condensed description of the view.	“VIEW-DEFN command” on page 99
VIEWS-FOR-USER	Lists the views related to the signed-on user along with the short text for the view.	“VIEWS-FOR-USER command” on page 102

Statistics commands

Command	Description	Section
PRINT-STATS	Prints the current statistics.	“PRINT-STATS command” on page 84
SHOW-NAVIGATION	Allows you to verify the accuracy of the access paths used by RDM to access the underlying entities during a view open.	“SHOW-NAVIGATION command” on page 87
STATS	Displays the current statistics for all open views or for a view you specify.	“STATS command” on page 90
STATS-OFF	Prints the current statistics and then disables the statistics gathering.	“STATS-OFF command” on page 91
STATS-ON	Initializes statistics to 0 and then enables the gathering of statistics on user views on both the logical and physical levels.	“STATS-ON command” on page 92

Signing on to DBAID

You sign on to DBAID by responding to the PLEASE SIGN ON prompt with your 1- to 30-character user name. If there is no password associated with your user name, simply press the RETURN key. However, you must supply a password if one was defined for you on the Directory. The password can be 1–8 alphanumeric or printable characters. You can enter the password either on the same line as the user name, or on a separate line. If you enter the password on the same line as the user name, you must precede the password with a space. If you enter the password on a separate line, it will not be echoed on-screen.

In the following example, INVENTORY-SYSTEM is the user name and PETER is the password. (Notice that a space precedes the password because it is on the same line as the user name.) When you successfully sign on, you receive the message SUCCESSFUL COMPLETION - SERVICE LEVEL *nn*. You can then begin entering commands.

```
                SUPRA SERVER RELEASE 2.n
                WELCOME TO DBAID - SERVICE LEVEL nn

PLEASE SIGN ON.
>INVENTORY-SYSTEM PETER
FSI: * VSI: = MSG: SUCCESSFUL COMPLETION - SERVICE LEVEL nn
```


Accessing the SUPRA Server HELP facility

SUPRA Server also provides a HELP facility. You can use HELP in two ways:

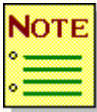
- ◆ Enter HELP in reply to a DBAID prompt. SUPRA Server displays a list of topics available within the HELP facility. You may then select the topic by entering enough of the topic name for the selection to be unique. Some topics also have a list of subtopics.
- ◆ Directly access a topic by entering the topic name with the command HELP as shown in the following example:

```
HELP DEL
```

This entry gives information specific to DELETE.



Note, however, to display information for the * command, you must use HELP ASTERISK. If you enter HELP *, the * acts as a wildcard. Wildcard means that all topics are included in the HELP description.



You can use the optional logical DBAID_HELP_NOSPAWN (set to TRUE) to call LBR\$OUTPUT_HELP instead of spawning a subprocess to display the HELP for DBAID.

*** command**

Use the asterisk character in the following two ways: (1) as a substitute for the last view name used or, (2) to denote a comment line by entering * in the first position after the prompt.

*

General consideration

Each section containing a command that supports * (as a substitute for the last view name used) describes how to use it.

Examples

- ◆ Using the * to denote a comment:

```
*ANY COMMENTS MAY BE PUT ON A DBAID LINE AFTER  
*AN ASTERISK IN FIELD 1.
```

- ◆ Using the * as a substitute for the last view name used:

```
OPEN VIEW  
GET *          (Performs GET on VIEW)  
OPEN VIEW2 = * column1,column5  
GET *          (Performs GET on VIEW2)
```

= command

Use = to perform an exact reissue of the previous RDML command.

=

General consideration

This command repeats an invalid command if the first key word is correct.

Example

In this example, = reissues the GET NEXT command preceding it.

```
GET NEXT CUST-PRODUCT-VIEW  
=
```

BYE command

Use BYE to exit the DBAID Test Facility.

BYE

General considerations

- ◆ BYE returns you to VAX/VMS processing.
- ◆ Any unsaved virtual views are erased.

BY-LEVEL command

Use BY-LEVEL to display the field names in a view by level of occurrence starting with the 0 level, followed by level 1, and so on. RDM generates the column number when displaying this data.

BY-LEVEL [*view-name* [*column-number*]]

view-name

Description *Optional.* Names the valid, open view whose column names you want to display.

Considerations

- ◆ If you omit this parameter, SUPRA Server displays all column names for all of your opened views.
- ◆ Entering * instead of a view name causes DBAID to substitute the last view name used.

column-number

Description *Optional.* Specifies the number of the field whose name is to be displayed.

Format Integer value.

Considerations

- ◆ If you use this parameter, you must specify a view name.
- ◆ If you omit this parameter, SUPRA Server displays all column names of the specified view.

Example This example displays the column names in the views CUST-PROD, CUSTOMER and TEST. They are listed by level of occurrence.

>BY-LEVEL NUMBER	VIEW NAME	FIELD NAME	LEVEL
1	CUST-PROD	CUST-NO	0
2	CUST-PROD	PROD-NO	1
3	CUST-PROD	RENT	1
4	CUST-PROD	MAINT	1
5	CUST-PROD	INSTALL-DATE	1
6	CUST-PROD	CANCEL-DATE	1
7	CUST-PROD	PURCHASE-PRICE	1
1	CUSTOMER	CUST-NO	0
2	CUSTOMER	NAME	0
3	CUSTOMER	STATE	0
1	TEST	ZONED5	1
2	TEST	PACKED5	1
3	TEST	KEY2	1

CAUTIOUS command

Use CAUTIOUS to disable the DBAID automatic COMMIT facility. This command is the opposite of SURE. When you use CAUTIOUS, DBAID does not automatically issue a COMMIT when an RDML INSERT, UPDATE or DELETE is issued. Instead, you must issue the COMMIT.

CAUTIOUS

General consideration

DBAID normally issues a COMMIT after every successful RDML modification. CAUTIOUS is not required; however, it gives you more control over COMMIT commands when updating the database.

COLUMN-DEFN command

Use COLUMN-DEFN to display the full description of columns in a view. For compatibility with previous releases of SUPRA Server, you can use **FIELD-DEFN** in the same manner as COLUMN-DEFN.

COLUMN-DEFN [*view-name* [*column-name*]]

view-name

Description *Optional.* Names the valid, open view to be used.

Considerations

- ◆ If you omit this parameter, COLUMN-DEFN displays all column descriptions for all of your opened views.
- ◆ Entering * instead of a *view-name* causes DBAID to substitute the last *view-name* used.

column-name

Description *Optional.* Identifies the column whose description you want to display.

Format The column must already be a part of the view.

Considerations

- ◆ If you use this parameter, you must have specified a *view-name*.
- ◆ If you omit this parameter, COLUMN-DEFN displays all column descriptors for each column of the specified view, one at a time.

Example

This example displays a description of the MANUAL-TITLE column in the REVIEW-DETAILS view. See the tables following this example for an explanation of each column descriptor.

VIEW-NAME	(+) REVIEW-DETAILS
COL-NAME	(+) MANUAL-TITLE
COL-POS	(+) 20
COL-LEN	(+) 30
COL-ASI-POS	(+) 109
COL-DEC	(+) 0
COL-OUTP-LEN	(+) 30
COL-MASK-LEN	(-) 0
COL-FORMAT	(+) C
COL-MASK	(-)
COL-HEADING	(-)
COL-DEL-OPT	(+) Y
COL-INS-OPT	(+) Y
COL-UPD-OPT	(+) Y
COL-REDUND	(+) N
COL-CONSTANT	(+) N
COL-LEVEL	(+) 1
COL-KEY-NUM	(+) 0
COL-REQUIRED	(+) Y
COL-UNIQUE	(+) Y
COL-EDIT-TRANS	(+)
COL-ORDERING	(-)
COL-SIGNED	(+) N
COL-NULLS-OK	(+) N
COL-NULL-LEN	(-) 0
COL-NULL-VAL	(-)
COL-DOMAIN	(-)
COL-VAL-TYP	(-)
COL-GET-VAL	(+) Y
COL-MIN-LEN	(-) 0
COL-MIN-VAL	(-)
COL-MAX-LEN	(-) 0
COL-MAX-VAL	(-)
COL-VAL-TABLE	(-)
COL-EXIT	(-)
COL-SRC-TYP	(+) F
COL-SRC-COL	(+) BOOK-REVIEWED
COL-SRC-REL	(+) BOREBOOK
COL-INT-REL	(+) BORE
COL-RC	(+)

Keys for the view

Column descriptor	Explanation
VIEW-NAME	Name of the view
COL-NAME	Name of the column

Data needed to read the column from the row

Column descriptor	Explanation
COL-POS	Offset of column value from start of row
COL-LEN	Length of column value
COL-ASI-POS	Distance ASI for column is offset from start of user buffer

Data needed to display the column

Column descriptor	Explanation
COL-DEC	Number of decimal places
COL-OUTP-LEN	Edited output length
COL-MASK-LEN	Length of output mask
COL-FORMAT	Column format
COL-MASK	Column mask
COL-HEADING	Column heading

Logical data about the column

Column descriptor	Explanation
COL-DEL-OPT	Y = Column may be deleted
COL-INS-OPT	Y = Column may be inserted
COL-UPD-OPT	Y = Column may be updated
COL-REDUND	Y = Column is redundant
COL-CONSTANT	Y = Column is a constant
COL-LEVEL	Level of occurrence
COL-KEY-NUM	0–9 = Column key number
COL-REQUIRED	Y = Column is required
COL-UNIQUE	Y = Column is unique
COL-EDIT-TRANS	Reserved for future use
COL-ORDERING	A = Ascending order, D = Descending order
COL-SIGNED	Y = Column is signed

Data about null value for the column

Column descriptor	Explanation
COL-NULLS-OK	Y = Nulls are allowed
COL-NULL-LEN	Length of the null value
COL-NULL-VAL	Null value in external format

Validation criteria for the column

Column descriptor	Explanation
COL-DOMAIN	Domain name, if any
COL-VAL-TYP	Validation type (R = Range, T = Table, E = Exit)
COL-GET-VAL	Y = Validation done after
GETCOL-MIN-LEN	Length of minimum value
COL-MIN-VAL	Minimum value in external format
COL-MAX-LEN	Length of maximum value
COL-MAX-VAL	Maximum value in external format
COL-VAL-TABLE	Validation table name
COL-EXIT	Validation exit name

Source column data

Column descriptor	Explanation
COL-SRC-TYP	Source type for the column (F = Data set, V = View).
COL-SRC-COL	If COL-SRC-TYPE is F, this field contains the logical data item name; if COL-SCR-TYPE is V, this field contains the source column name.
COL-SRC-REL	If COL-SRC-TYPE is F, this field contains the physical data item name; if COL-SRC-TYPE is V, this field contains the source view name.
COL-INT-REL	If COL-SCR-TYPE is F, this field contains the data set name; if COL-SRC-TYPE is V, this field contains the user view name.
COL-RC	If COL-SRC-TYPE is F, this field contains the record code, if any.

COLUMN-TEXT command

Use COLUMN-TEXT to display the comments for a column in a view. For compatibility with previous releases of SUPRA Server, you can use **FIELD-TEXT** in the same manner as COLUMN-TEXT.

COLUMN-TEXT [*view-name* [*column-name*]]

view-name

Description *Optional.* Names the valid, open view.

Considerations

- ◆ If you omit this parameter, the COLUMN-TEXT command displays the short and long text for all of your opened views.
- ◆ Entering * instead of a *view-name* causes DBAID to substitute the last *view-name* used.

column-name

Description *Optional.* Identifies the column whose text you want to display.

Format The column must already be a part of the view.

Considerations

- ◆ If you use this parameter, you must specify a *view-name*.
- ◆ If you omit this parameter, COLUMN-TEXT displays the comments for all columns.

COMMIT command

Use COMMIT to issue an RDM COMMIT request. All updates since the last COMMIT are made permanent in the database.

COMMIT

General consideration

DBAID automatically issues a COMMIT after every successful modification. COMMIT is not required in DBAID.

DELETE command

Use DELETE to issue an RDM DELETE request, which removes a view-record occurrence from the database.

DELETE [ALL] *view-name*

ALL

Description *Optional.* Deletes all rows that satisfy the logical-key qualification of the **GET** issued before the DELETE.

Consideration If a program specifies a GET without a USING phrase, DELETE ALL deletes all rows in a view.

view-name

Description *Required.* Names the valid, opened view that contains the record(s) you want to delete.

Considerations

- ◆ Before performing the DELETE you should perform a successful GET that contains a FOR UPDATE clause, in case the record changes between the GET and the DELETE.
- ◆ You can enter * instead of a *view-name*. This causes DBAID to substitute the last *view-name* used.

General considerations

- ◆ RDM deletes records only if the ALLOW clause specifies DEL or ALL. (See the fourth item in the following examples.)
- ◆ RMS interfile integrity is only maintained for those files within the view. RDM does not check indexes to other files not included in the Access Set.

Examples

- ◆ This example deletes one occurrence of SAMPLE-VIEW obtained by using the value in KEY1:

```
GET SAMPLE-VIEW FOR UPDATE USING KEY1
DELETE SAMPLE-VIEW
```

- ◆ This example deletes all occurrences of rows:

```
GET SAMPLE-VIEW FOR UPDATE USING KEY1
DELETE ALL SAMPLE-VIEW
```



The above example works as if the following loop were performed:

```
GET FIRST SAMPLE-VIEW FOR UPDATE USING KEY1

NOT FOUND GO TO CONTINUE.

LOOP.

GET NEXT SAMPLE-VIEW FOR UPDATE USING KEY1

NOT FOUND GO TO CONTINUE.

DELETE SAMPLE-VIEW.

GO TO LOOP.

CONTINUE.
```

- ◆ This example deletes all rows in SAMPLE-VIEW:

```
GET SAMPLE-VIEW.
DELETE ALL SAMPLE-VIEW.
```

- ◆ This RMS example shows the statements used by the **GET** and **DELETE** commands to delete all rows from the CUST and ORDR files:

```
NAME:    CUST-ORDER-VIEW
KEY CUST-NO
KEY ORDR-NO
KEY ORDR-PROD-NO
ACCESS CUST USING CUST-NO ALLOW DEL INS
ACCESS ORDR USING (CUST-NO, ORDR-PROD-NO) ALLOW ALL
GET CUST-ORDER-VIEW
DELETE ALL CUST-ORDER-VIEW
```

- ◆ This RMS example shows the statements used by the **GET** and **DELETE** for a deletion using an alternate index. The physical key **ORDR- PROD-NO** is the alternate index for **PRODNKEY**:

```
NAME:      CUST-ORDER-PROD-VIEW
KEY CUST-NO
KEY ORDR-NO
KEY ORDR-PROD-NO
      PROD DESC
ACCESS CUST USING CUST-NO
ACCESS ORDR VIA PRODNKEY
      USING (CUST-NO, ORDR-NO, ORDR-PROD-NO) ALLOW ALL
ACCESS PROD USING ORDR-PROD-NO ALLOW ALL
GET CUST-ORDER-PROD-VIEW
DELETE CUST-ORDER-PROD-VIEW
```

ERASE command

Use ERASE to cause DBAID to issue an RDM RESET if an X FSI is returned from an RDML command. This command is the opposite of **KEEP**.

ERASE

FIELD-DEFN command

Use FIELD-DEFN to display the full description of columns in a view.

FIELD-DEFN [*view-name* [*column-name*]]

view-name

Description *Optional.* Specifies the view to be used.

Considerations

- ◆ If you omit this parameter, SUPRA Server displays all column descriptions for all of your opened views.
- ◆ Entering * instead of a *view-name* causes DBAID to substitute the last *view-name* used.

column-name

Description *Optional.* Identifies the column whose description is to be displayed.

Format The column must already be a part of the view.

Considerations

- ◆ If you use this parameter, you must have specified a *view-name*.
- ◆ If you omit this parameter, SUPRA Server displays all column descriptors for each column of the specified view, one at a time.

Example This example displays the full description of all columns in all open views. See the table following this example for an explanation of each column descriptor.

```

> FIELD-DEFN
VIEW-NAME                (+) CUSTOMER
FIELD-NAME                (+) CUST-NO
FIELD-POS                 (+) 0
FIELD-LEN                 (+) 5
ASI-POS                   (+) 60
FIELD-DEC                 (+) 0
OUTPUT-LEN                (+) 5
MASK-LEN                  (+) 15
FORMAT                    (+) Z
EDIT-MASK                  (+) ZZZZZZZZZZZZZ9
HEADING                   (+) CUST;NO
DELETABLE                 (+) Y
INSERTABLE                (+) Y
REPLACABLE                (+) N
FIELD-LVL                 (+) 0
KEY-NUMBER                (+) 1
REQUIRED                  (+) Y
UNIQUE                    (+) Y
EDIT-TRANS                (+) E
ORDERING                  (-)
***MORE***

```

Column descriptor	Explanation
VIEW-NAME	The name of the view being described.
FIELD-NAME	The name of the column being described.
FIELD-POS	The position of the column in the user's buffer, starting at byte 0.
FIELD-LEN	The length of the column.
ASI-POS	The position of the ASI of this column in the user buffer.
FIELD-DEC	The number of decimal places in the column.
OUTPUT-LEN	The length of the output column.
MASK-LEN	The length of the edit mask.
FORMAT	The format of the column.
EDIT-MASK	Not implemented for this release.
HEADING	Not implemented for this release.
DELETABLE	Indicates whether the record may be deleted.
INSERTABLE	Indicates whether the record may be inserted.
REPLACEABLE	Indicates whether this column may be updated.
FIELD-LVL	Indicates the level of the record which contains this column.
KEY-NUMBER	Indicates which key column this is; 0 indicates the column is not a key, 1 is the first key, and so on, up to 9.
REQUIRED	Indicates the column must not be null when performing updates or inserts.
UNIQUE	Indicates the column is a unique key.
EDIT-TRANS	Not implemented for this release.
ORDERING	Indicates a linkpath is ordered using this column.

FIELD-TEXT command

Use FIELD-TEXT to display the comments for a column in a view.

FIELD-TEXT [*view-name* [*column-name*]]

view-name

Description *Optional.* Names the valid, opened view.

Format Must be a valid and opened view.

Considerations

- ◆ If you omit this parameter, SUPRA Server displays all column descriptions for all of your opened views.
- ◆ Entering * instead of a *view-name* causes DBAID to substitute the last *view-name* used.

column-name

Description *Optional.* Identifies the column whose text is to be displayed.

Format The column must already be part of the view.

Considerations

- ◆ If you use this parameter, you must specify a *view-name*.
- ◆ If you omit this parameter, the comments for all columns are displayed.

Example This example displays the comments for all columns in all open views:

```
> FIELD-TEXT
      VIEW NAME                                FIELD NAME
-----
L15-A3                                A-CTRL
-----
                                COMMENTS
-----
FIRST COMMENT
SECOND COMMENT
-----
***MORE***
      VIEW NAME                                FIELD NAME
-----
L15-A3                                A-DATA
-----
NO COMMENT
----- ***MORE***
```

FORGET command

Use FORGET to remove the specific mark from the list of marks in use and frees the storage allocated by a previous **MARK**.

FORGET *mark-name*

mark-name

Description *Required.* Specifies what mark information should be forgotten.

Format 1–30 alphanumeric characters

Consideration Must be a name you assigned with MARK.

General consideration

Once you issue a FORGET, the indicated mark is released and cannot be used without issuing a new MARK.

GET command

Use GET to retrieve and display a row for the indicated view.

	[<u>NEXT</u>]	
	[LAST]	
GET	[SAME]	<i>view – name</i>
	[FIRST]	
	[PRIOR]	

[FOR UPDATE]

AT *mark-name*

USING *literal*₁ [*literal*₂ *literal*₃ ...*literal*_{*n*}]

NEXT
LAST
SAME
FIRST
PRIOR

Description *Optional.* Modifies the order of retrieval of rows.

Default NEXT If no current position exists, NEXT defaults to FIRST.

Considerations

- ◆ For a unique key:
 - NEXT Retrieves either the row immediately after the current row or the first row if no current position exists.
 - LAST Retrieves the last row.
 - SAME Retrieves the latest row if a current position exists.
 - FIRST Retrieves the first row in the view.
 - PRIOR Retrieves either the row immediately before the current row or the last row if no current position exists. Use GET PRIOR only in connection with a USING key phrase for predictable results.
 - GET
PRIOR
and GET
LAST Cannot be performed on primary data sets. An error is returned.
- ◆ For a nonunique key:
 - NEXT Retrieves the next occurrence of the row within the generic group.
 - LAST Retrieves the last occurrence of the row.
 - SAME Retrieves the latest row if a current position exists.
 - FIRST Retrieves the first occurrence of the row with the indicated key.
 - PRIOR Performs a read reverse within the group of nonunique keyed records.
- ◆ For RMS, GET LAST and GET PRIOR are not supported. If you want to access base RMS data sets, supply the key.

view-name

Description *Required.* Names the valid, opened view to be used.

Consideration Entering * instead of a *view-name* causes DBAID to substitute the last *view-name* used.

FOR UPDATE

Description *Optional.* Allows you to lock out other users' modifications to the row you are retrieving.

Considerations

- ◆ The FOR UPDATE phrase allows you to perform modifications dependent upon the current contents of the row.
 - ◆ If you do not need to be certain of the content of the row, use GET without the FOR UPDATE phrase. When the **UPDATE** or **DELETE** function is performed, the Automatic Hold facility performs the lock before modifying the record.
 - ◆ FOR UPDATE locks all physical resources until you issue another GET or an **INSERT**, **UPDATE**, **DELETE**, **COMMIT**, or **RESET**. This practice might lead to system inefficiency.
-

AT *mark-name*

Description *Optional.* Repositions a view previously marked with **MARK**.

Consideration USING and AT may not be used with the same GET command.

USING *literal*₁ [*literal*₂ *literal*₃ ... *literal*_n]

Description *Optional.* Identifies a value or set of values for a keyed GET.

Format The values must be part of a valid view. Separate the items with a space and use either character, hexadecimal, or numeric data. Character and hexadecimal data must be enclosed in quotes; numeric data does not. For example:

Using 'ABCD' - Character data

Using X'A10C' - Hexadecimal

Using 1234 - Numeric data

Using 123 'ABC' - Combination (two keys)

Considerations

- ◆ The number of keys specified in the GET must be less than or equal to the number of keys in your specified attribute list. No more than nine keys are allowed in one view.
- ◆ Any omitted keys are treated as generic keys. The use of generic keys is a convenient feature for allowing both direct access to a view and a sequential scan of many records. All occurrences of a particular unspecified column are returned as long as the other keys are satisfied.
- ◆ The order of specified keys in the USING phrase corresponds to the order of key declarations in your attribute list.

Examples

- ◆ This example returns the row whose key is JONES from the view LASTNAME. No position is specified, so NEXT is assumed. However, since this is the first GET, the first (or only) record with this key is returned:

```
GET LASTNAME USING 'JONES'
```

- ◆ This example returns the next row with the key JONES. When you have read all JONES records, you get the message OCCURRENCE NOT FOUND:

```
GET * USING 'JONES'
```

- ◆ This example returns the first JONES with the initials APQ:

```
GET * USING 'JONES' 'APQ'
```

- ◆ This example uses the default NEXT. The next record is returned (the record following JONES APQ). You can use this command repeatedly to get all records in a view, such as the rest of the Joneses, then all Robinsons, then all Smiths, and so on:

```
GET *
```

- ◆ In this RMS example, GET retrieves data from the ORDR file using the file physical key:

```
GET FIRST ORDER-VIEW USING 225600x100
```

- ◆ This RMS example uses the view ORDER-VIEW with GET to retrieve data from the ORDR file:

```
NAME: ORDER-VIEW
KEY FULL-ORDR-KEY
    ORDR-SALE-TOTAL
    ORDR-DATE-OF-SALE
ACCESS ORDR USING FULL-ORDR-KEY
GET ORDER-VIEW
```

- ◆ In this RMS example, the physical key of the file ORDR consists of three subparts. A generic read is performed when the application program only supplies a value for ORDR-CUST-NO or a combination of ORDR- CUST-NO and ORDR-NO on the GET:

```
GET ORDER-VIEW USING CUST01
GET ORDER-VIEW USING CUST01 ORDR02
```

- ◆ All products for the requested order are returned when the values for CUST-NO and ORDR-NO are supplied. The GET command uses the full physical key if all three logical keys are supplied:

```
GET ORDER-VIEW USING CUST01
                        ORDR02
                        PART11
```

- ◆ This RMS example is similar to the preceding example. However, ORDR- PROD-NO is left out of the USING phrase to force a generic read of the ORDR file:

```
GET ORDER-VIEW USING CUST02
```

You can leave fields out of the USING clause from the right. RDM generic reads can be performed only from within RDM based on the logical key supplied within the USING clause in the ACCESS statements.

GO command

Use GO to issue a GET request based on a single key followed by a series of sweeping GET requests scanning one-to-many relationships (the records are displayed in a tabular format).

```
GO [NEXT
   PRIOR] view - name

[START {NEXT
        LAST
        SAME
        FIRST
        PRIOR
        AT mark - name}]

[FOR number - of - rows]

[FROM literal1 [literal2...literaln]
  USING
```

[NEXT
PRIOR]

Description	Optional. Specifies the positional modifier for subsequent retrievals after the initial access by GET.
Default	NEXT

view-name

Description	Required. Names the valid, opened view to be accessed.
Consideration	Entering * instead of a view-name causes DBAID to substitute the last view-name used.

```
[ START { NEXT  
        LAST  
        SAME  
        FIRST  
        PRIOR  
        AT mark - name } ]
```

Description *Optional.* Specifies GET positional modifier for the initial access of the database.

Default FIRST If GO NEXT is specified

 PRIOR If GO PRIOR is specified

FOR *number-of-rows*

Description *Optional.* Indicates the number of rows (or number of GET NEXTs minus 1) to be performed.

Default 16,777,216

Format Numeric characters

Consideration GET NEXTs will be issued until the count is exhausted or until the last row is retrieved, whichever occurs first.

```
[FROM  literal1 [literal2...literaln] ]  
[USING
```

Description	<i>Optional.</i> Identifies a value or set of values used for a keyed GET.	
Format	Either character or numeric data. Character data, if it includes blanks, must be enclosed in quotes; numeric data does not.	
Options	FROM	The key values are used only on the initial access; the scan is unqualified.
	USING	The key values are used for both the initial access and the subsequent scan.

Considerations

- ◆ The number of keys specified in GET must be less than or equal to the number of keys in your specified attribute list.
- ◆ Any omitted keys are treated as generic keys. The use of generic keys allows for both direct access to a view and a sequential scan of many records. All occurrences of a particular unspecified key are returned as long as the other keys are satisfied.
- ◆ The order of specified keys in the USING phrase corresponds to the order of key declarations in your attribute list.

General considerations

- ◆ The output is displayed in columns. If more data is to be displayed than fits on a screen, an alternate format is used.
- ◆ After GO displays a page of records (see “**PAGESIZE command**” on page 83), the *****MORE***** prompt is issued. You may continue the display on the next page after input of a blank line.
- ◆ At the end of the series of rows retrieved by GO, the *****END***** prompt is issued.
- ◆ “For number-of-records” is not recommended for online use because it does not pause until the last screen.
- ◆ GO always looks ahead one record so it can determine whether to display the *****MORE***** or *****END***** message. It can be confusing if you issue a GET after the GO because a record might appear to have been skipped.

Examples

- ◆ The command GO VIEW START AT VIEW-MARK1 USING (VIEW-KEY-VALUE) issues the following sequence of RDM GET commands until a not-found FSI is returned:

```
GET VIEW AT VIEW-MARK1
GET NEXT VIEW USING (VIEW-KEY-VALUE)
GET NEXT VIEW USING (VIEW-KEY-VALUE)
.
.
.
```

- ◆ The command GO PRIOR VIEW START LAST FROM (VIEW-KEY-VALUE) issues the following sequence of RDM GET commands until a not-found FSI is returned:

```
GET LAST VIEW USING (VIEW-KEY-VALUE)
GET PRIOR VIEW
GET PRIOR VIEW
.
.
.
```

INSERT command

Use INSERT to issue an RDM INSERT request. The INSERT places a view record in the physical database based on the relative location specified.

```
INSERT [NEXT  
      LAST  
      FIRST  
      PRIOR] view - name [MASS]
```

- NEXT
- LAST
- FIRST
- PRIOR

Description	Optional. Specifies where the record will be inserted in relation to existing rows. The Access Set Description (ASD) may override this specification.	
Default	NEXT	If not positioned in the view, NEXT defaults to LAST, and PRIOR defaults to FIRST.

Considerations For nonuniquely keyed values:

- ◆ **INSERT FIRST.** Places a row in the first position in the view.
- ◆ **INSERT NEXT.** Places a row after the current row. If no current position exists, the row is placed in the last position in the view.
- ◆ **INSERT PRIOR.** Places a row before the current row. If no current position exists, the row is placed in the first position in the view.
- ◆ **INSERT LAST.** Places a row in the last position of the view.

view-name

Description *Required.* Names the valid, opened view in which you want the rows inserted.

Considerations

- ◆ Entering * instead of a *view-name* causes DBAID to substitute the last *view-name* used.
- ◆ After you enter the column values, the row is displayed. The message INSERT (Y/N) displays, and you must respond accordingly.

MASS

Description *Optional.* Inserts many rows.

Considerations

- ◆ The positioning parameter you specify is used by RDM on every INSERT command issued by mass insert.
- ◆ Input logical records immediately following this command after the prompts MASS INSERT PROCESSING INITIATED and ENTER END. TO EXIT MASS INSERT.
- ◆ Records are inserted as flat records. Separate the columns with commas. Insert rows longer than one line by terminating the list of values with a comma.
- ◆ If columns have no values, enter two consecutive commas to indicate their absence. This value is treated as a null value for packed or zoned items, as a large number for binary items, and as blanks for character items.
- ◆ If columns contain single quotes (apostrophes), replace them with two single quotes (not double quotes) and enclose the entire string in single quotes. If columns contain spaces, enclose the entire string in single quotes.
- ◆ Specify END. after you have input all rows to be inserted into the view.
- ◆ To place multiple records on a single line, leave a blank between rows. Do not specify the view name while doing a mass insert.
- ◆ Processing stops if 10 errors are detected while using MASS insert; otherwise, enter END. to terminate inserting.

General considerations

- ◆ If you use INSERT without MASS, SUPRA Server prompts you for values even if the view does not allow inserts.
- ◆ Quotes may be used to include blanks in character strings.

Examples

The following examples use INSERT in an online environment. The fields after the > prompt indicates user input.

- ◆ This example inserts a view record in the physical database:

```
>INSERT *  
NUMBER  
>9998  
PRODUCT  
>AAAA  
INSTALLED  
>100883  
NUMBER                               ( ) 9998  
PRODUCT                             ( ) AAAA  
INSTALLED                           ( ) 100883  
INSERT (Y/N)?  
>Y  
FSI: * VSI: + MSG: SUCCESSFUL COMPLETION
```

- ◆ These examples use MASS to insert one or more rows without reference to column names. You use a blank to indicate the end of an inserted row. You can also enter one or more records per line, using a comma to carry part of a record to the next line. You use END. to stop mass inserting:

a.

```
>INSERT + MASS
MASS INSERT PROCESSING INITIATED.
ENTER "END." TO EXIT MASS INSERT.
>9997,BBBB,100783
FSI: * VSI + MSG: SUCCESSFUL COMPLETION
```

b.

```
>9996,CCCC,
>100683
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
<9995,DDDD,100583,9994,EEEE,100483 9993,FFFF,100383
FSI * VIS: + MSG: SUCCESSFUL COMPLETION
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
FSI * VSI: + MSG: SUCCESSFUL COMPLETION
>END.
MASS INSERT PROCESSING COMPLETED.
```

- ◆ This example shows an insert to an RMS data set:

```
INSERT CUST-ORDER-PROD-VIEW
```

- ◆ In this RMS example, the following view is used with INSERT. An insert is attempted on each file. If at least one record is inserted, the operation is successful.

```
NAME: CUST-ORDER-PROD-VIEW
KEY CUST-NO
CUST-NAME
KEY ORDR-NO
KEY ORDR-PROD-NO
PROD-DESC
ACCESS CUST USING CUST-NO ALLOW ALL
ACCESS ORDR USING (CUST-NO, ORDR-NO, ORDR-PROD-NO) ALLOW ALL
ACCESS PROD USING ORDR-PROD-NO ALLOW ALL
```

- ◆ This RMS example shows how to prohibit an insert on a particular view by not coding INS or ALL on the ALLOW clause. In this example, whether or not the PROD record exists, the ORDR record is not inserted. If the record exists, a message indicating that an invalid value is in a required field appears, and an ASI of V is returned on the key fields:

```
NAME:      CUST-ORDER-VIEW
KEY  CUST-NO
      CUST-NAME
KEY  ORDR-NO
KEY  ORDR-PROD-NO
ACCESS CUST USING CUST-NO ALLOW ALL
ACCESS ORDR USING (CUST-NO, ORDR-NO, ORDR-PROD-NO) ALLOW ALL
ACCESS PROD USING ORDR-PROD-NO ALLOW DEL
```

KEEP command

Use KEEP to disable the DBAID automatic RESET facility. This command is the opposite of **ERASE**. KEEP prohibits DBAID from issuing a RESET when it receives an X FSI from the view. Instead, DBAID “keeps” the database as it is and allows the user to decide whether to RESET or not. This is the default setting.

KEEP

LINE SIZE command

Use LINE SIZE to display the number of characters in a line or the current line-size setting.

LINE SIZE [*number-of-characters*]

number-of-characters

Description *Optional.* Indicates the number of characters to be displayed on a line.

Default 77

Options 11–256

Consideration If you omit the number-of-characters, the command displays the current LINE SIZE setting.

MARK command

Use MARK to mark the current position of the view record established by the previous **GET**.

MARK *view-name* **AT** *mark-name*

view-name

- Description** *Required.* Identifies the *view-name* established by the previous GET.
- Format** Must be a valid and opened view.
- Consideration** Entering * instead of a *view-name* causes DBAID to substitute the last *view-name* used.

AT *mark-name*

- Description** *Required.* Names the location where the position of the current view will be marked.
- Format** 1–30 alphanumeric characters.
- Consideration** You use this name in a later GET AT request to retrieve this view record.

General considerations

- ◆ The AT clause in GET repositions the view at the position set by the MARK.
- ◆ You may create any number of marks for a logical user view, but to conserve space, reuse MARKs when possible.

Example This example marks the current position of the view record:

```
>MARK CUSTOMER AT REMEMBER-CUSTOMER
```

You can do other GETs on CUSTOMER and return to this mark immediately.

MARKS command

Use MARKS to list all open MARKs and the views they are marking.

MARKS

Example

This example lists all open marks and the views (CUST-PROD) they are marking:

>MARKS	
MARK NAME	VIEW NAME
MARK6	CUST-PROD
MARK5	CUST-PROD
MARK4	CUST-PROD
MARK3	CUST-PROD

OPEN command

Use OPEN to ready a saved or virtual view for use by DBAID.

OPEN [*user-view-name*=] *view-name*[*column*₁,...,*column*_{*n*}]

user-view-name=

Description *Optional.* Gives an existing view a name to be used in DBAID.

Format 1–30 alphanumeric characters. The first character must be alphabetic.

Considerations

- ◆ If *user-view-name* is not specified, the *view-name* is used.
- ◆ Use this command with the column parameter to create many smaller views from one common view.
- ◆ To OPEN a view that has not been listed or defined in the same session of DBAID, the user must be related to the view in the Directory.

view-name

Description *Required.* Names the virtual or stored view to be readied for use.

Format Must be a valid view.

Considerations

- ◆ Entering * instead of a *view-name* causes DBAID to substitute the last *view-name* used.
- ◆ If you have Access or Privileged Database Administrator authority , you may use LIST to list any view before you open it. This makes the text of the view known to DBAID. Such a view is called a virtual view. Issuing OPEN on a view without first issuing a LIST causes RDM to directly open the view before making text available to DBAID. Refer to the *SUPRA Server PDM Database Administration Guide (UNIX & VMS)*, P25-2260, for information on LIST.

column₁,...,column_n

Description *Optional.* Identifies the column(s) to be included in the user view.

Format The columns must already be a part of the view being opened.

Considerations

- ◆ You may continue the list of column names on successive lines by ending the current line with a comma. This will be necessary if the current line size is less than the space required to enter all of the data items in the row.
- ◆ USER-LIST displays the list of columns used to open the view (see “**USER-LIST command**” on page 98).

General consideration

OPEN returns information about the storage used in the message:

```
nnnnn BYTES USED IN OPENING VIEW
```

where *nnnnn* is the amount of storage used by the view.

Example This example opens the user view CP-ONLY. The view comprises a subset of all of the columns in CUST-PROD:

```
>OPEN CP-ONLY = CUST-PROD CUST-NO,PROD-NO
```

Only CUST-NO and PROD-NO are returned by GET CP-ONLY, even though CUST-PROD has six defined columns.

PAGESIZE command

Use PAGESIZE to specify the number of lines to be displayed on a screen/page or to display the current page-size setting.

PAGESIZE [*number-of-lines*]

number-of-lines

Description *Optional.* Indicates the number of lines to be displayed on a screen/page.

Format Two or more numeric characters greater than 10.

Consideration If you omit *number-of-lines*, the command displays the current page-size setting.

General consideration

The initial page size is 24 lines.

PRINT-STATS command

Use PRINT-STATS to cause RDM to print logical and physical statistics such as the number of **GET**s, **INSERT**s, and **DELETE**s done on a view. You may issue the command numerous times during a session after you have first issued **STATS-ON**.

PRINT-STATS

General considerations

- ◆ STATS-ON must precede the first PRINT-STATS; if you do not first issue STATS-ON, PRINT-STATS has no effect.
- ◆ You may issue **STATS-OFF** to discontinue statistics gathering. **BYE** and **SIGN-OFF** print statistics and then turn off statistics gathering.
- ◆ Use PRINT-STATS to keep a statistical running total.

Example

In the following example, PRINT-STATS prints statistics after each RDML operation:

```
STATS-ON
GET NEXT CUST-PROD
.
.
PRINT-STATS
UPDATE CUST-PROD
.
.
PRINT-STATS
```

RELEASE command

Use RELEASE to issue an RDM RELEASE command that closes a specific view or all opened views and releases the occupied storage.

RELEASE [*view-name*]

view-name

Description *Optional.* Names the valid, opened view to be released.

Considerations

- ◆ Entering * instead of *view-name* causes DBAID to substitute the last *view-name* used.
- ◆ If you omit this parameter, all of your opened views are released.

General considerations

- ◆ The definition of any view is retained, allowing subsequent retrieval and processing.
- ◆ This command does not affect virtual view text of the view(s).

RESET command

Use RESET to issue an RDM RESET request. A RESET rolls back any database updates for the current user since the last **COMMIT**.

RESET

General considerations

- ◆ Only use RESET after unsuccessful updates. DBAID issues a COMMIT after every successful update.
- ◆ DBAID does not automatically issue a RESET when an X FSI is returned.

SHOW-NAVIGATION command

Use SHOW-NAVIGATION to verify the accuracy of the access paths used by RDM to access the underlying entities during a view open.

SHOW-NAVIGATION [*view-name*]

view-name

Description *Optional.* Names the valid, opened view for which you wish to display details of access paths used.

Considerations

- ◆ Entering * instead of *view-name* causes DBAID to return information on the *view-name* used most recently.
- ◆ If you omit the *view-name* parameter, RDM returns information on all opened views in turn.

Example The following example shows the access path using the view REGION-BY-NAME. REGION-BY-NAME is a base view because it accesses a data set, REGN, through the secondary index key REGNSKNM.

```
>SHOW-NAVIGATION REGION-BY-NAME
```

SIGN-OFF command

Use SIGN-OFF to sign off from DBAID.

SIGN-OFF

General consideration

Use SIGN-OFF to remove yourself as a user without terminating DBAID.

SIGN-ON command

Use SIGN-ON to identify yourself to DBAID.

SIGN-ON *username* [*password*]

username

Description	<i>Required.</i> Names the user.
Format	1–30 alphanumeric characters. Must be a valid user name already defined on the Directory.

password

Description	<i>Conditional.</i> Required if a password is associated with the user name. However, if no password is associated with it, then press RETURN after the password prompt.
Format	1–8 alphanumeric characters. Must be a valid password already defined on the Directory.

General consideration

A SIGN-ON is done on entry to DBAID.

Example This example identifies Jane Doe to DBAID:

```
>SIGN-ON JDOE DBAPSWD
```

STATS command

Use STATS to cause RDM to display the current statistics for all open views or for a view that you specify.

STATS [*view-name*]

view-name

- Description**
- Optional.* Names the valid, opened view for which you wish to display statistics.
- Consideration**
- Entering an * instead of a *view-name* causes DBAID to substitute the last *view-name* used.

General considerations

- ◆ STATS-ON must precede the first STATS; if you do not first issue STATS-ON, STATS has no effect.
- ◆ You may issue a STATS-OFF to discontinue statistics gathering.
- ◆ When you issue STATS, the statistics are displayed on your screen.
- ◆ You may issue STATS-OFF followed by STATS-ON, or just STATS-ON to reset the statistical information.
- ◆ Use STATS to keep a statistical running total.

Example

The following example uses STATS to display a running total after each RDML operation:

```
STATS-ON
GET NEXT REVIEW-DETAILS
.
.
.
STATS
UPDATE REVIEW-DETAILS
.
.
STATS
```

STATS-OFF command

Use STATS-OFF to cause RDM to print the current statistics. After RDM prints the statistics, it displays them.

STATS-OFF

General considerations

- ◆ STATS-ON must precede STATS-OFF.
- ◆ Issuing STATS-OFF without a preceding STATS-ON has no effect.
- ◆ BYE or SIGN-OFF will also perform a STATS-OFF.

STATS-ON command

Use STATS-ON to cause RDM to initialize the statistics to 0 and then begin gathering statistics. The DBA can use this command, together with **STATS-OFF** or **PRINT-STATS**, to examine what user views do on both logical and physical levels.

STATS-ON

General considerations

- ◆ Statistics are gathered on a task basis, not on a systemwide basis.
- ◆ Use STATS-OFF to print statistics and then turn them off.
- ◆ Use PRINT-STATS to print statistics but continue gathering a running total.
- ◆ You can use **BYE** and SIGN-OFF to print statistics and then turn them off.

SURE command

Use SURE to cause a COMMIT after each successful INSERT, UPDATE, or DELETE. SURE is the opposite of CAUTIOUS and will cause RDM to automatically issue a COMMIT if an * FSI is returned by a RDML command that alters the database. SURE is the default setting.

SURE

UNDEFINE command

Use UNDEFINE to remove the name and definition of a virtual view.

```
UNDEFINE { ALL
          view - name }
```

ALL

view-name

Description	<i>Required.</i> Specifies which virtual views to remove.	
Options	ALL	Removes all virtual views currently in use and issues an RDM RELEASE.
	view-name	Identifies the virtual view to be removed. This must be a valid view.

Consideration Entering * instead of *view-name* causes DBAID to substitute the last *view-name* used.

General considerations

- ◆ The view relinquishes the storage, allowing it to be reclaimed for defining other views.
- ◆ This command does not remove a saved definition from the Directory.

Examples

- ◆ This example removes all views currently in use:
 >UNDEFINE ALL
- ◆ This example removes the view CUSTOMER:
 >UNDEFINE CUSTOMER

UPDATE command

Use UPDATE to update data values in the database. For RMS data sets, it also updates the relationships between files.

UPDATE *view-name*

[*column₁:=literal₁* [, ..., *column_n:=literal_n*]]

view-name

Description *Required.* Names the valid, opened view you wish to update.

Consideration Entering an * instead of a *view-name* causes DBAID to substitute the last *view-name* used.

column₁:=literal₁[,...,column_n:=literal_n]

Description	<i>Optional.</i>	Identifies a column in the view which is to have the value of the literal.
Format	<i>column</i>	The column must already be part of the view being updated.
	<i>:=</i>	Must be coded as shown.
	<i>literal</i>	Character or numeric data. Hexadecimal value is not allowed.

Considerations

- ◆ Each updateable column is displayed, and replacement values are accepted. Entering a null line does not change the item; entering new data changes the item value in the row. After all updateable items are processed, the UPDATE (Y/N) prompt is displayed and requires a response.
- ◆ You can use the column:=literal syntax when updating items in the row. Only the items you specify are updated; all others remain the same. To update a row, indicate the item you want to update, the :=, and the new value for the item.
- ◆ Single quotes are not required around a character or numeric literal unless the literal contains spaces or commas.
- ◆ Single quotes are required for you to change the value of a column to blanks. A literal of spaces (keyed in) must be in single quotes. If you press RETURN, you do not affect the item's value.
- ◆ UPDATE cannot be used to modify columns designated as key values. Use DELETE and INSERT to modify key items. See “DELETE command” on page 53 and “INSERT command” on page 72.
- ◆ To UPDATE a row, you must first retrieve the row using the GET command.
- ◆ UPDATE cannot change all the values in a defined column to a specific value. For example, you cannot change all PROD-CODES to T100.
- ◆ If the physical field being updated is an alternate key for RMS data sets, RDM maintains the secondary index in the same file as the primary index.

Examples

- ◆ This example updates the columns RENT and MAINT in the view CUST-PROD:


```
>UPDATE CUST-PROD RENT:=175.00, MAINT:=50.00
```
- ◆ This RMS example uses the following view with UPDATE to enter customer, product, and date of sale information within the data set:


```
NAME: ORDER-DATE-VIEW
KEY  ORDR-CUST-NO
KEY  ORDR-PROD-NO
      ORDR-DATE-OF-SALE
ACCESS ORDR USING (ORDR-CUST-NO, ORDR-PROD-NO) ALLOW ALL
ACCESS DATE USING ORDR-DATE-OF-SALE ALLOW ALL
>GET ORDER-DATE-VIEW
>UPDATE ORDER-DATE-VIEW ORDR-DATE-OF-SALE:=19NOV85
```
- ◆ If an alternate index is defined for the above view, RDM performs the update as follows:
 - Before the record is deleted in the DATE file, a check is made through the alternate index to see if any records in the ORDR file have the old value.
 - If so, the delete is not performed on the DATE file. RDM then attempts to insert the new value into the DATE file.
 - If a duplicate occurrence is found, the error is ignored.

USER-LIST command

Use USER-LIST to display the attribute list for the user view named.

USER-LIST *user-view-name*

user-view-name

- Description** *Required.* Identifies the user view or view to be displayed.
- Format** Must be a valid user view.
- Consideration** Entering * instead of a *user-view-name* causes DBAID to substitute the last *user-view-name* used.
- Example** This example displays the list of attributes for the PO-CODE-ONLY user view:
- ```
>USER-LIST PO-CODE-ONLY
USER VIEW NAME : PO-CODE-ONLY
VIEW VIEW NAME : CUSTOMER-PURCHASE-ORDER
USER VIEW LIST :
CUST-NO , PURCHASE-ORDER-CODE , END .
```

---

# VIEW-DEFN command

Use VIEW-DEFN to display a condensed description of a view.

---

## VIEW-DEFN [*view-name*]

---

---

### *view-name*

- Description**     *Optional.* Specifies the view whose condensed description is to be displayed.
- Format**            Must be a valid and opened view.

**Considerations**

- ◆ Entering \* instead of a *view-name* causes DBAID to substitute the last *view-name* used.
- ◆ If you omit this parameter, a condensed description of all your opened views is displayed.

**Example**            This example displays a condensed description of the CUSTOMER view. The table following this example explains each displayed descriptor.

```
> VIEW-DEFN
VIEW-NAME (+) CUSTOMER
INS-ORDER (+) N
TOTAL-SIZE (+) 63
TOTAL-FIELDS (+) 3
TOTAL-LEVELS (+) 1
TOTAL-DELETABLE (+) 3
TOTAL-INSERTABLE (+) 3
TOTAL-REPLACABLE (+) 3
TOTAL-REQUIRED (+) 1
TOTAL-KEYS (+) 1
TOTAL-NONUNIQUE (+) 0
MORE
```

| View descriptor   | Explanation                                                                 |
|-------------------|-----------------------------------------------------------------------------|
| VIEW-NAME         | The name of the view being described.                                       |
| INS-ORDER         | Indicates that inserts will be ordered, depending on the value of a column. |
| TOTAL-SIZE        | The total number of bytes in the view, including ASIs.                      |
| TOTAL-FIELDS      | The number of columns in the view.                                          |
| TOTAL-LEVELS      | The number of levels in the view.                                           |
| TOTAL-DELETABLE   | The number of deletable columns.                                            |
| TOTAL-INSERTABLE  | The number of insertable columns.                                           |
| TOTAL-REPLACEABLE | The number of updateable columns.                                           |
| TOTAL-REQUIRED    | The number of required columns.                                             |
| TOTAL-KEYS        | The number of keys in the view.                                             |
| TOTAL-NONUNIQUE   | The number of nonunique keys in the view.                                   |

---

## VIEWS command

Use VIEWS to display all of the views currently active in DBAID.

---

### VIEWS

---

#### General consideration

The information displayed with this command includes:

- ◆ **User View.** The name of the user view.
- ◆ **View.** The name of the view from which the user view is taken.
- ◆ **Status.** Indicates whether the user view is open or released.

#### Example

This example displays all views currently active in DBAID:

```
>VIEWS
USER VIEW VIEW
STATUS
CUSTOMER-PURCHASE-ORDER CUSTOMER-PURCHASE-ORDER
OPENED
PO-CODE-ONLY CUSTOMER-PURCHASE-ORDER
OPENED
```

# VIEWS-FOR-USER command

Use VIEWS-FOR-USER to list the views related to the signed-on user.

## VIEWS-FOR-USER

### Example

This example displays the views related to the signed-on user:

| VIEWS-FOR-USER |                    |   |          |
|----------------|--------------------|---|----------|
| !              | LOGICAL VIEW NAME  | ! | DATE     |
| !              | TIME               | ! |          |
| !              | -----!             | ! | -----!   |
| !              | BASE-VIEW          | ! | 03/03/95 |
| !              | REGION             | ! | 14:22:18 |
| !              | BRANCH             | ! | 03/03/95 |
| !              | CUSTOMER           | ! | 14:12:43 |
| !              | PRODUCT            | ! | 04/05/95 |
| !              | BRANCH-SUBSET      | ! | 10:40:19 |
| !              | BRANCHES-IN-REGION | ! | 16:42:56 |
| !              | PRODUCTS-IN-REGION | ! | 03/03/95 |
| !              | REVIEW-DETAILS     | ! | 10:40:29 |
| !              | WRITING-DETAILS    | ! | 04/25/95 |
| !              | MANUALS            | ! | 16:39:40 |
| !              | AUTHOR             | ! | 03/30/95 |
| !              | PRODUCTION-DETAILS | ! | 14:33:38 |
| !              | -----!             | ! | 15:17:41 |
| !              |                    | ! | 10:34:21 |
| !              |                    | ! | 15:14:29 |
| !              |                    | ! | 15:16:05 |
| !              |                    | ! | 15:16:36 |
| !              |                    | ! | 15:17:23 |
| !              | -----!             | ! | -----!   |

# 4

## Writing an RDM program in COBOL, FORTRAN, and BASIC (VMS)

Use the information in this chapter to write RDM programs in COBOL, FORTRAN, and BASIC. The statements (and their associated optional phrases) presented here and in “[Coding RDM program statements \(VMS\)](#)” on page 153 are written as part of the source language. The RDML preprocessor converts the RDML statements into the correct set of MOVEs and CALLs to the RDM processor. Each RDML statement must start on a line by itself.

### Understanding RDML statement format

The following are format considerations for RDML statements:

#### COBOL

- ◆ You may use either ANSI or terminal format COBOL.
- ◆ Use dashes, not underscores, when coding data names. The COBOL preprocessor does not accept underscores.
- ◆ End each statement with a period.

## **FORTRAN and BASIC**

- ◆ Do not use a period at the end of any RDML statements.
- ◆ Remember that any comment line is also a comment for the preprocessor. End of line comments are permitted in RDML source wherever they are permitted in FORTRAN or BASIC. In FORTRAN, these comments are preceded by a C; in BASIC, by an exclamation point (!).
- ◆ If you need to extend an RDML statement over several source lines, then use the FORTRAN or BASIC system for continuation lines. Characters after column 72 are ignored.
- ◆ In FORTRAN, the preprocessor includes debug lines when counting the number of statements in the output file.
- ◆ In BASIC, all preprocessor statements are preceded by the keyword RDM, except for DUP KEY, NOT FOUND, and END ERROR-HANDLER. For compatibility with ULTRA 1.4, you may also use the keyword LUV.
- ◆ Do not use the following words as user view names because they conflict with the syntax of the INCLUDE and the various ACCESS statements: COMMON, FIRST, LAST, PRIOR, NEXT, SAME, and ALL. If you use these words, you will get compile errors.
- ◆ Use hyphens and underscores as follows:
  - FORTRAN. All names defined on the Directory are stored in COBOL format; hyphens are allowed, as well as alphanumerics. FORTRAN allows only underscores and the dollar sign (\$), as well as alphanumerics. The two systems are reconciled by replacing hyphens with underscores when conversion to FORTRAN is required. The preprocessor replaces both underscores and dollar signs with hyphens when the reverse transformation is required.
  - BASIC . Names defined on the Directory can contain hyphens. To refer to such a name in your program, code an underscore instead of a hyphen.



- ◆ If you are using FORTRAN, you must use FORTRAN format names within FORTRAN RDML statements with the exception of the INCLUDE statement. In the INCLUDE statement, you can use either FORTRAN or COBOL format names, because the INCLUDE statement is regarded as the interface between the Directory and the FORTRAN compiler.
- ◆ If you are using BASIC, do not embed RDM statements in other statements. These two examples are invalid:

```
IF I=1 THEN RDM INSERT V1
RDM INSERT V1 UNLESS I < > 1
```

In addition, they must be separated from other statements by new lines or by a /. For example:

```
IF I=1 THEN / RDM INSERT / END IF
```

The lines in the program file must be in sequential order (all line numbers in the file must be in order).

The following three tables give the valid combinations of data type, length, and decimal places for COBOL, FORTRAN, and BASIC data items. Note that some combinations valid for COBOL are invalid for FORTRAN.

When using the information in the Description Column, consider the following:

- ◆ Length of the data item (L)
- ◆ Number of decimal places (D)
- ◆ Total number of digits, both before and after the decimal point (B)

In addition, an X in the Signed/Unsigned column means it is irrelevant.

| <b>Data type</b> | <b>Signed / Unsigned</b> | <b>Length</b> | <b>Decimal places</b> | <b>COBOL description</b> |
|------------------|--------------------------|---------------|-----------------------|--------------------------|
| Character        | X                        | 1–9999        | 0                     | PIC X(L)                 |
| Binary           | S                        | 1             | 0                     | PIC X                    |
| Binary           | U                        | 1             | 0                     | PIC X                    |
| Binary           | S                        | 2             | 0                     | PIC S9(4) COMP           |
| Binary           | U                        | 2             | 0                     | PIC 9(4) COMP            |
| Binary           | S                        | 4             | 0                     | PIC S9(9) COMP           |
| Binary           | U                        | 4             | 0                     | PIC 9(9) COMP            |
| Binary           | S                        | 8             | 0                     | PIC S9(18) COMP          |
| Binary           | U                        | 8             | 0                     | PIC 9(18) COMP           |
| Binary           | S                        | 2             | 1–4                   | PIC S9(B)V9(D) COMP      |
| Binary           | U                        | 2             | 1–4                   | PIC 9(B)V9(D) COMP       |
| Binary           | S                        | 4             | 1–9                   | PIC S9(B)V9(D) COMP      |
| Binary           | U                        | 4             | 1–9                   | PIC 9(B)V9(D) COMP       |
| Binary           | S                        | 8             | 1–18                  | PIC S9(B)V9(D) COMP      |
| Binary           | U                        | 8             | 1–18                  | PIC 9(B)V9(D) COMP       |
| Numeric          | S                        | 1–18          | 0                     | PIC S9(B)                |
| Numeric          | U                        | 1–18          | 0                     | PIC 9(B)                 |
| Numeric          | S                        | 1–18          | 1-L                   | PIC S9(B)V9(D)           |
| Numeric          | U                        | 1–18          | 1-L                   | PIC 9(B)V9(D)            |
| Packed decimal   | S                        | 1–10          | 0                     | PIC 9(L) COMP-3          |
| Packed decimal   | S                        | 1–10          | 1 to (2L-1)           | PIC 9(B)V9(D) COMP-3     |
| Floating point   | S                        | 4             | 0                     | COMP-1                   |
| Floating point   | S                        | 8             | 0                     | COMP-2                   |

| <b>Data type</b> | <b>Signed / Unsigned</b> | <b>Length</b> | <b>Decimal places</b> | <b>FORTTRAN description</b> |
|------------------|--------------------------|---------------|-----------------------|-----------------------------|
| Character        |                          | 1–9999        | 0                     | CHARACTER*L                 |
| Binary           | S/U                      | 2             | 0                     | INTEGER*2                   |
| Binary           | S/U                      | 4             | 0                     | INTEGER*4                   |
| Floating point   | S/U                      | 4             | 0                     | REAL                        |
| Floating point   | S/U                      | 8             | 0                     | DOUBLE PRECISION            |

| <b>Data type</b> | <b>Signed / Unsigned</b> | <b>Length</b> | <b>Decimal places</b> | <b>BASIC description</b> |
|------------------|--------------------------|---------------|-----------------------|--------------------------|
| Character        | X                        | 1–9999        | 0                     | STRINGX=L                |
| Binary           | S/U                      | 1             | 0                     | BYTE                     |
| Binary           | S/U                      | 2             | 0                     | WORD                     |
| Binary           | S/U                      | 4             | 0                     | LONG                     |
| Packed decimal   | S                        | 1–10          | 0                     | DECIMAL (B,D)            |
| Packed decimal   | S                        | 1–10          | 1 to (2L-1)           | DECIMAL (B,D)            |
| Floating point   | S                        | 4             | 0                     | SINGLE                   |
| Floating point   | S                        | 8             | 0                     | DOUBLE                   |

## Enrolling your program in the SUPRA directory

Use the following information to enroll your program in the SUPRA Directory.

### Writing the identification division (COBOL)

Write the identification division as you would for any COBOL program, for example:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. LVPROG.
```

Include the program name since it is used to enroll the program on the Directory.

### Writing the program naming statement (FORTRAN and BASIC)

In FORTRAN, write the Program Naming statement as you would write it for any program. For example:

```
PROGRAM LVDEMO
```

This statement is optional in VAX FORTRAN; however, it must be included in a program using RDML so that the SUPRA Directory can enroll your program.

In BASIC, write the Program Naming statement as follows:

```
RDM PROGRAM LVDEMO
```

---

## Defining program data

When you define program data, you describe the information that your program will process. This includes specifying the input/output records and their data files. It also includes:

- ◆ Specifying views, user views, and ULT-CONTROL
- ◆ Validating your data by checking status indicators
- ◆ Specifying how RDM checks that you are using a current program

### Writing the environment division (COBOL)

If you are using COBOL, first write the environment division as you would for any COBOL program, for example:

```
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
 SELECT INPUT-FILE ASSIGN TO READER.
 SELECT OUTPUT-FILE ASSIGN TO DISK.
```

The FILE-CONTROL statements are required only if your program will be using files other than PDM data sets.

## Writing the Data Division (COBOL) and the Declaration Statements (FORTRAN and BASIC)

Using the COBOL Data Division or the FORTRAN and BASIC Declaration Statements, you specify views, user views and a special view called ULT-CONTROL; validate your data by checking status indicators; and define how RDM checks that you are using a current program.

### Specifying views and user views

You can use a view that is already established or create your own view.

To use a view that your DBA established, code an INCLUDE with the name of the view. For example:

#### ◆ COBOL

```
01 INCLUDE PART-COMP.
```

#### ◆ FORTRAN

```
PROGRAM LVDEMO
INCLUDE PART-COMP
```

#### ◆ BASIC

```
RDM PROGRAM LVDEMO
RDM INCLUDE PART-COMP
```

You may create your own user view by selecting columns from a view. To create a user view, code an INCLUDE with the name of the subset of the view, the name of the view, and the columns you want to include in the view. For example:

#### ◆ COBOL

```
01 V2 INCLUDE PART-COMP (PART=PART-NAME,COMP=COMPONENT-NAME)
```

#### ◆ FORTRAN

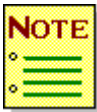
```
INCLUDE PART-COMP=V2 (PART=PART-NAME,COMP=COMPONENT-NAME)
```

#### ◆ BASIC

```
RDM INCLUDE PART-COMP=V2 (PART=PART-NAME,COMP=COMPONENT-NAME)
```

If you create your own user view by identifying subsets of columns, RDM carries out the validation that has been defined for the complete view (you cannot change the effect of required columns in terms of their being available for inserts). For example, if an INSERT is executed and a column is required for the addition of that row, but the column is not defined in your subset, RDM rejects the INSERT. You can also modify the order of columns in a view. However, if you reorder logical keys in a view, you may inadvertently force RDM to read many rows from the view to satisfy the GET if you did not specify a physical key value. For each INCLUDE, the RDM preprocessor generates associated record and status data areas. The record data area specifies where data for each included view is placed in the program. The status data area directly corresponds to the columns in the record data area. It also contains 1 byte of information indicating whether the data is valid, has changed since your most recent access, or is missing (see “Validating data” on page 115).

The preprocessor changes RDML statements into comments. These comments are preceded by an asterisk (\*) in COBOL, a C in FORTRAN, and an exclamation point (!) in BASIC. At this point, the expanded version of the program is ready for compiling, as illustrated below.




---

Never change the expanded (precompiled) code; only change the code before it is precompiled.

---

## COBOL

```
*01 CUSTOMER INCLUDE CUST (CUST-NO,NAME,CITY) .
01 LUV-CUSTOMER.
 10 CUSTOMER.
 20 CUST-NO PIC X(006) .
 20 NAME PIC X(020) .
 20 CITY PIC X(015) .
 10 ASI-CUSTOMER.
 20 ASI-CUST-NO PIC X .
 20 ASI-NAME PIC X .
 20 ASI-CITY PIC X .
```

**FORTRAN**

```

C INCLUDE PART-COMP=V2 (PART=PART-NAME, COMP=COMPONENT-NAME)
CHARACTER*6 PART
CHARACTER*6 COMP
CHARACTER*1 ASI_PART, ASI_COMP
EQUIVALENCE (PART, PART_COMP(1:6))
EQUIVALENCE (COMP, PART_COMP(7:12))
EQUIVALENCE (ASI_PART, PART_COMP(13:13))
EQUIVALENCE (ASI_COMP, PART_COMP(14:14))
INTEGER*4 PART_COMP_LEN
PARAMETER(PART_COMP_LEN=14)
CHARACTER*(PART_COMP_LEN) PART_COMP
CHARACTER ULT$PART_COMP*30, ULT$PART*17, ULT$COMP*22, ULT_END_VIEW1*4
DATA ULT$PART_COMP/'V2' /ULT$PART
+/'006C00PART-NAME,'/ULT$COMP/'006C00COMPONENT-NAME,/ULT_END_VIEW
+1/'END.'/'
CHARACTER *73 ULT$1
EQUIVALENCE (ULT$1, ULT$PART_COMP), (ULT$1(31:47), ULT$PART),
+(ULT$1(48:69), ULT$COMP), (ULT$1(70:73), ULT_END_VIEW1)
+

```

**BASIC**

```

! RDM INCLUDE FRED=TEST6D(F1=CUST-ORDER-NO, F2=CUST-ORDER-DATE)
RECORD FRED_REC
STRING F1=6, STRING F2=9
STRING ASI_F1 = 1, ASI_F2 = 1
END RECORD
DECLARE FRED_REC FRED
DECLARE STRING CONSTANT FRED_ULT = 'TEST6D'
"006C00CUST-ORDER-NO, "+"009C00CUST-ORDER-DATE, "+"END."
' +

```



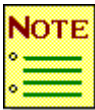
## Specifying ULT-CONTROL

Include the special view ULT-CONTROL in each program issuing an RDML request. ULT-CONTROL contains operation, status, and other control-oriented information required to control access to all views. It passes parameters between the application and the RDM.

When you specify an INCLUDE for ULT-CONTROL, RDM generates the following:

### COBOL

```
*01 INCLUDE ULT-CONTROL.
01 ULT-CONTROL.
 10 ULT-OBJECT-NAME PIC X(30).
 10 ULT-OPERATION.
 20 ULT-ID PIC X(2).
 20 ULT-OPCODE PIC X.
 20 ULT-POSITION PIC X.
 20 ULT-MODE PIC X.
 20 ULT-KEYS PIC X.
 10 ULT-FSI PIC X.
 10 ULT-VSI PIC X.
 10 FILLER PIC X(2).
 10 ULT-MESSAGE PIC X(40).
 10 ULT-PASSWORD PIC X(8).
 10 ULT-OPTIONS PIC X(4).
 10 ULT-CONTEXT PIC X(4).
 10 ULT-LVCONTEXT PIC X(4).
```




---

You can use an INCLUDE in either the Working-Storage or Linkage Section.

---

**FORTRAN**

```

C INCLUDE ULT-CONTROL
 CHARACTER
 ULT_OBJECT_NAME*30,ULT_OPERATION*6,ULT_FSI*1,ULT_VSI*1,
 +ULT_FILLER*2,ULT_MESSAGE*40,ULT_PASSWORD*8,ULT_OPTIONS*4,
 +ULT_CONTEXT*4,ULT_LVCONTEXT*4
 PARAMETER(ULT_CONTROL_LEN=100)
 CHARACTER*(ULT_CONTROL_LEN) ULT_CONTROL
 EQUIVALENCE (ULT_CONTROL(1:30),ULT_OBJECT_NAME(1:30)),
 +(ULT_CONTROL(31:36),ULT_OPERATION(1:6)),(ULT_CONTROL(37:37),
 +ULT_FSI(1:1)),(ULT_CONTROL(38:38),ULT_VSI(1:1)),
 +(ULT_CONTROL(39:40),ULT_FILLER(1:2)),(ULT_CONTROL(41:80),
 +ULT_MESSAGE(1:40)),(ULT_CONTROL(81:88),ULT_PASSWORD(1:8)),
 +(ULT_CONTROL(89:92),ULT_OPTIONS(1:4)),(ULT_CONTROL(93:96),
 +ULT_CONTEXT(1:4)),(ULT_CONTROL(97:100),ULT_LVCONTEXT(1:4))
 CHARACTER*14 ULT_DATE_STAMP
 DATA ULT_DATA_STAMP/'19831114143849'/
C ON ERROR
C TYPE *, 'RDM control call failed',ULT_FSI
C STOP
C END ERROR-HANDLER
C

```

**BASIC**

```

! RDM INCLUDE ULT-CONTROL
RECORD ULT_CONTROL_REC
STRING ULT_OBJECT_NAME = 30, ULT_OPERATION = 6,
 ULT_FSI = 1,ULT_VSI = 1, ULT_FILLER = 2,ULT_MESSAGE = 40,
 ULT_PASSWORD = 8, ULT_OPTIONS = 4,ULT_CONTEXT = 4, ULT_LVCONTEXT
 = 4
END RECORD
DECLARE ULT_CONTROL_REC ULT_CONTROL
EXTERNAL SUB CSVIPLVS, CSVBERROR
DECLARE STRING CONSTANT ULT_DATE_STAMP = "19840830161953"
! RDM ON ERROR
! GOTO 99
! END ERROR-HANDLER

```

## Validating data

The DBA's choice of data type for a column restricts the values you can insert into the database. When you run a program, RDM rejects invalid values.

RDM returns three kinds of status indicators to the application program to indicate RDML processing results.

- ◆ **FSI (Function Status Indicator).** Indicates the success or failure of the function and is returned after any RDML function call.
- ◆ **ASI (Attribute (Column) Status Indicator).** Indicates the status of each column in the row and is returned after a DELETE, INSERT, GET, or UPDATE RDML function call.
- ◆ **VSI (Validity Status Indicator).** Indicates the validity of the user view row returned by your last RDML request and is returned after a DELETE, INSERT, GET, or UPDATE RDML function call.

Function Status Indicators

Function Status Indicators (FSI) reflect the success or failure of your RDML request. RDM obtains the 1-character field (ULT-FSI) from ULT-CONTROL and provides an associated message in ULT-CONTROL's message area (see the example in "Specifying ULT-CONTROL" on page 113). The RDML processor returns only one FSI for any RDML function. FSIs have the following meanings:

| FSI value | Meaning                                                                                                                                                                                                        |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| *         | Successful.                                                                                                                                                                                                    |
| D         | Data error. Row contains invalid or changed data (VSI=C). Check the ASI to find the column(s) containing the invalid value.                                                                                    |
| F         | Fail. Indicates a major error. Something may be wrong with the database, or you might have attempted to perform an invalid function on the user view.                                                          |
| N         | Fail due to occurrence problem. This may be because of a GET not found or an INSERT duplicate found.                                                                                                           |
| R         | DYNAMIC RESET. The PDM has performed a dynamic reset on the database because of an earlier PDM failure. The PDM has restarted automatically, so you must reapply all modifications made since the last COMMIT. |
| S         | Security. Verify the RDML function and correct if necessary.                                                                                                                                                   |
| U         | Unavailable resources. The resource required to complete this function was not available; retry at a later time.                                                                                               |
| X         | RDML function modifications were made to the database before the error condition was detected. Issue a RESET to restore the database. This code overrides D, F, S, or U indicators.                            |

## Attribute (Column) Status Indicators

Attribute (Column) Status Indicators (ASI) reflect the status of each column in your view. The RDML processor returns one ASI for each column in the user view and places it in the ASI element-name fields defined for you by the RDML preprocessor.

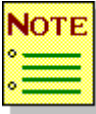
Use ASIs to:

- ◆ Determine which columns have null values or which have been changed since the most recent access through the current view.
- ◆ Find which columns have invalid values if your program receives an FSI indicating a data error.
- ◆ (COBOL only) Avoid run-time errors when your program uses a view which has packed values. Run-time errors are caused if your program performs a calculation or move using an invalid packed decimal value. Examine each ASI before performing a move or calculation, taking note of the following points:
  - If the ASI is V, the value is placed in the row even though it is not in a valid format.
  - If the ASI is -, the column value is a valid 0 value for numeric columns or the column value is spaces for character columns.
  - For other ASI values, the column is valid.
- ◆ Request RDM to set a column to its null value on INSERTS and UPDATES.

You may access the ASIs through names generated by the RDML compiler. You must code an INCLUDE for the user view in your program. The RDML preprocessor generates:

- ◆ A statement for each column included in the user view row.
- ◆ A column for each ASI by preceding the name of each column in the view with the characters ASI- (COBOL) or ASI\_ (FORTRAN and BASIC). The RDML preprocessor truncates any trailing characters in a column name with over 26 characters.

The following are examples of this generation:



The ASIs are defined after the last column in the user view.

## COBOL

The asterisk indicates the statement you code. The RDML compiler generates all other statements.

```
*01 PROD1 INCLUDE PRODUCT.
01 LUV-PROD1.
10 PROD1.
 20 PROD-NO PIC X(004).
 20 PROD-DESC PIC X(040).
10 ASI-PROD1.
 20 ASI-PROD-NO PIC X.
 20 ASI-PROD-DESC PIC X.
```

## FORTRAN

The C indicates the statement you code. The RDML compiler generates all other statements.

```
C INCLUDE PART-COMP=V2 (PART=PART-NAME , COMP=COMPONENT-NAME)
CHARACTER*6 PART
CHARACTER*6 COMP
CHARACTER*1 ASI_PART,ASI_COMP
EQUIVALENCE (PART,PART_COMP(1:6))
EQUIVALENCE (COMP,PART_COMP(7:12))
EQUIVALENCE (ASI_PART,PART_COMP(13:13))
EQUIVALENCE (ASI_COMP,PART_COMP(14:14))
INTEGER*4 PART_COMP_LEN
PARAMETER (PART_COMP_LEN=14)
CHARACTER*(PART_COMP_LEN)PART_COMP
CHARACTER ULT$PART*30,ULT$PART*17,ULT$COMP*22,ULT_END_VIEW1*4
DATA ULT$PART_COMP/'V2' /ULT$PART
+/'006C00PART-NAME','/ULT$COMP/'006C00COMPONENT-NAME',/ULT_END_VIEW
+1/'END.' /
CHARACTER *73 ULT$1
EQUIVALENCE (ULT$1,ULT$PART_COMP), (ULT$1(31:47),ULT$PART),
+(ULT$1(48:69),ULT$COMP), (ULT$1(70:73),ULT_END_VIEW1)
+
```

**BASIC**

The ! indicates the statement you code. The RDML compiler generates all other statements.

```
! RDM INCLUDE FRED=TEST6D(F1=CUST-ORDER-NO,F2=CUST-ORDER-DATE)
RECORD FRED_REC
STRING F1=6,STRING F2=9
STRING ASI_F1 = 1,ASI_F2 = 1
END RECORD
DECLARE FRED_REC FRED
DECLARE STRING CONSTANT FRED_ULT = 'TEST6D' +
"006C00CUST-ORDER-NO,"+"009C00CUST-ORDER-DATE,"+"END."
```

ASIs have the following meanings:

| ASI value | Meaning                                                                                                                          |
|-----------|----------------------------------------------------------------------------------------------------------------------------------|
| C         | The column value was changed by another view.                                                                                    |
| V         | The value of the column is not a valid value.                                                                                    |
| -         | The column is missing or has a null value.                                                                                       |
| +         | The column exists but another task has changed it since the most recent access.                                                  |
| =         | The column exists and has not been changed by another task since the most recent access.                                         |
| N         | You can place an N in the ASI during UPDATES and INSERTS to set a column to its null value. RDM never returns an ASI value of N. |

Validity Status Indicators

Validity Status Indicators (VSI) reflect the validity of the user view row returned by your most recent RDML request. The RDML processor returns the VSI to the program in an area generated as part of the programmer-supplied INCLUDE ULT-CONTROL statement (see the example in “Specifying ULT-CONTROL” on page 113).

You can use this indicator to determine the most significant ASI returned by RDM according to the following hierarchy:

| VSI value | Meaning                                                                                                        |
|-----------|----------------------------------------------------------------------------------------------------------------|
| C         | At least one column value was changed by another view.                                                         |
| V         | At least one invalid ASI was returned.                                                                         |
| -         | No invalid ASIs were returned, but at least one missing ASI was returned.                                      |
| +         | No invalid or missing ASIs were returned, but at least one column in the row has been changed by another task. |
| =         | No invalid, missing, or new physical occurrences were returned by this RDM function.                           |

The VSI enables you to determine if any additional processing of ASIs is needed to correct invalid data or to fill in missing values.



## Checking for current program

RDM keeps track of when columns change, when programs are compiled, what versions are used, and whether or not they are still compatible. When you execute an application program, RDM ensures that the program is still executable according to the current Directory definition. If the program is not current, RDM does not allow it to execute.

RDM checks to see if any columns in the view have been modified in the SUPRA Directory, SUPRAD, since the most recent program compile. If the DBA has made such a change, your program will receive the message CURRENCY CHECK - PLEASE RECOMPILE and cannot access the view until it has been recompiled. Changes which make recompiling necessary are:

- ◆ Data type change (packed decimal to binary, etc.)
- ◆ Deleted column (if the column is not part of your user view, you do not need to recompile)
- ◆ Column length change
- ◆ (COBOL only) Change in the number of decimal places

## Defining program logic

Write the procedure division (COBOL) and/or program logic statements (FORTRAN and BASIC) so you can perform the following:

- ◆ Sign On/Off
- ◆ Retrieve Rows
- ◆ Modify Rows
- ◆ Control Database Recovery
- ◆ Handle Error Conditions

### Signing on/off

Use SIGN-ON to establish communication between your program and RDM. SIGN-ON must be the first RDML command that your program executes. You must supply a user name and a password with the SIGN-ON if your DBA assigned you a password on the Directory. At run time, RDM checks the Directory to make sure the user name (and password, if necessary) is valid. By calling the program CSV\_SETUP\_REALM prior to sign-on, you may specify the mode in which each data set should be opened. See the General considerations in “SIGN-ON” on page 205 for details.

Use SIGN-OFF to terminate your session. RDM releases the storage areas acquired. Issue a SIGN-OFF at the end of every application program. See “SIGN-OFF” on page 202 for more information about SIGN-OFF.

## Retrieving rows

Use GET to retrieve rows with or without keys. You can also retrieve multiple rows and hold rows for subsequent UPDATE.

### Retrieving rows without a key

Use GET to retrieve rows without a key. For example, by repeatedly issuing the request, GET CUST-INFO (use an underscore if you are using FORTRAN or BASIC), you can sequentially retrieve every row in the CUST-INFO view. You can also use GET FIRST, GET NEXT, GET PRIOR, GET LAST, and GET SAME to retrieve rows without a key.

If your DBA defines views without any logical keys for performing direct reads, the USING phrase (which indicates what key values to access) is invalid and causes an error.

### Retrieving rows with keys

By specifying key values with the USING clause of GET you indicate the values RDM should use to access the view. Each logical key value restricts the set of rows that RDM retrieves from the view. If more than one row satisfies the condition required by the specified logical key values, you can use GET NEXT to retrieve occurrences after the first. For example, the view ACCOUNT\_DATA has two keys, CUSTOMER\_NO and ACCOUNT\_NO. Together, they uniquely identify a row in the view. The following statement retrieves the first row in the view, ACCOUNT\_DATA.

#### COBOL

```
GET FIRST ACCOUNT-DATA.
```

#### FORTRAN

```
GET FIRST ACCOUNT_DATA
```

#### BASIC

```
RDM GET FIRST ACCOUNT_DATA
```

GET NEXT retrieves the next row.

## COBOL

```
GET NEXT ACCOUNT-DATA.
```

## FORTRAN

```
GET NEXT ACCOUNT_DATA
```

## BASIC

```
RDM GET NEXT ACCOUNT_DATA
```

GET NEXT retrieves the first row in a user view if no current position exists (if you issue no other GETs). GET SAME retrieves the same row as the previous GET; GET PRIOR retrieves the previous row; and GET LAST retrieves the last row.

The effect of GET NEXT when using key values which differ from those used in the previous GET NEXT is unpredictable. RDM may require a database scan to find the row required. With GET NEXT, this scan is performed relative to the current position; therefore, the key values required may be missed because they lie at positions prior to the current position. The result would be a NOT FOUND error. Therefore, use GET FIRST or GET LAST when you are uncertain of the current position within your data.

The effect of GET NEXT when a current position exists (but for different key values than specified on this GET NEXT) is predictable. However, it depends on the underlying view and PDM. Specifically, RDM requires a database scan to find the row specified by the key values supplied; then some rows might not be found because the starting position is the current row. Therefore, if you are uncertain what the current position is, use GET FIRST and GET LAST.

After RDM retrieves the last user row, it raises a NOT FOUND condition. Therefore, ensure that the logic of your program indicates what to do. For example:

### COBOL

```
GET NEXT ACCOUNT-DATA
NOT FOUND GO TO ACCOUNT-NOT-FOUND.
```

### FORTRAN

```
GET NEXT ACCOUNT_DATA
NOT FOUND
 GO TO 900
END IF
```

### BASIC

```
RDM GET NEXT ACCOUNT_DATA
NOT FOUND
 GO TO 900
END IF
```

You may retrieve user rows using a single key:

### COBOL

```
MOVE 71560 TO CUSTOMER-NO
GET ACCOUNT-DATA USING CUSTOMER-NO
```

or

```
MOVE 71560 TO WORK-FIELD
GET ACCOUNT-DATA USING WORK-FIELD
```

### FORTRAN

```
GET ACCOUNT_DATA USING CUSTOMER_NO
```

or

```
CUSTOMER_NO=71560
GET ACCOUNT_DATA USING CUSTOMER_NO
```

### BASIC

```
RDM GET ACCOUNT_DATA USING CUSTOMER_NO
```

or

```
CUSTOMER_NO=71560
RDM GET ACCOUNT_DATA USING CUSTOMER_NO
```

This statement retrieves subsequent rows with the same account number:

### COBOL

```
GET NEXT ACCOUNT-DATA USING CUSTOMER-NO.
NOT FOUND GO TO
NO-MORE-ACCOUNTS-FOR-CUSTOMER
```

### FORTRAN

```
RDM GET NEXT ACCOUNT_DATA USING CUSTOMER_NO
NOT FOUND
 GO TO 900
END IF
```

### BASIC

```
GET NEXT ACCOUNT_DATA USING CUSTOMER_NO
NOT FOUND
 GO TO 900
```

If you use END IF and you specify enough logical keys to identify a row uniquely in a view, the second GET always returns a NOT FOUND condition. Only use the following statement once, because there should be only one such row:

### COBOL

```
GET ACCOUNT-DATA USING CUSTOMER-NO
 ACCOUNT-NO.
```

### FORTRAN

```
GET ACCOUNT_DATA USING CUSTOMER_NO
 ACCOUNT_NO
```

### BASIC

```
RDM GET ACCOUNT_DATA USING CUSTOMER_NO,ACCOUNT_NO
```

## Retrieving rows using partial keys

If you are accessing a row using a secondary key, you may omit characters from the right, substituting the wildcard characters \* or =. Note that you may have substituted your own wild card characters for the defaults. This is referred to as a generic read.

To illustrate a generic read, assume you are using a view, CUST-LOCATION, containing the columns NAME and CITY. A series of GETs would return data in the following order:

| NAME       | CITY       |
|------------|------------|
| ADAMS A    | ABERDEEN   |
| BROWN D    | READING    |
| CARSON C   | ABERDEEN   |
| COX D      | READING    |
| LOVE C     | PORTSMOUTH |
| SMITH Fred | SLOUGH     |
| SMITH P    | LONDON     |
| SMITH SM   | MAIDENHEAD |
| SMYTH M    | SWINDON    |

The wildcard \* specifies an equal or next match. RDM returns a row with a key that matches the partial key you specified. If no key matches, RDM returns the next row. The next value depends on the sort direction the DBA defined when creating the secondary key. RDM always returns a row for this option until it reaches the end of the file marker.

The following table uses the information in the NAME and CITY columns above and shows the order in which RDM retrieves the information when you issue a GET using the wildcard \*:

| Statement                      | Retrieval order       |            |
|--------------------------------|-----------------------|------------|
| GET CUST-<br>LOCATION USING C* | CARSON C              | ABERDEEN   |
|                                | COX D                 | READING    |
|                                | LOVE C                | PORTSMOUTH |
|                                | SMITH Fred            | SLOUGH     |
|                                | SMITH P               | LONDON     |
|                                | SMITH SM              | MAIDENHEAD |
|                                | SMYTH M               | SWINDON    |
|                                | OCCURRENCE NOT FOUND. |            |

The wildcard = specifies an equal match. RDM returns only those rows whose keys exactly match the partial key supplied. The following table uses the information in the NAME and CITY columns on the previous page and shows the order in which RDM retrieves the information when you issue a GET using the wildcard =:

| Statement                      | Retrieval order       |          |
|--------------------------------|-----------------------|----------|
| GET CUST-<br>LOCATION USING C= | CARSON C              | ABERDEEN |
|                                | COX D                 | READING  |
|                                | OCCURRENCE NOT FOUND. |          |

When RDM cannot find a row to match the partial key supplied, it returns OCCURRENCE NOT FOUND.



## Considerations for generic reads

- ◆ You must access the data set containing the partial key via a secondary key. Use DBAID SHOW-NAVIGATION to find out which access path RDM is using to obtain the data (see [“Using the DBAID Utility subset to test views \(VMS\)”](#) on page 35).
- ◆ The partial key specified must have a character data type. Generic read does not work on the other data types supported by RDM.
- ◆ The wildcard character must be the rightmost character in the supplied generic key (you can omit parts of the key from the right only). RDM ignores any values entered after the wildcard character. For example:

```
GET (view-name) USING SM* is valid
```

```
GET (view-name) USING SM*TH is accepted but is treated as SM*
```

- ◆ You can use compound generic keys; however, the wildcard character must still be the rightmost character in the string. For example, for a secondary key with two key parts:

```
Correct usage: GET (view-name) USING 1234 TE=
```

```
Incorrect usage: GET (view-name) USING 12= TEMP
```

- ◆ You can specify logical keys after the generic key. For example, for an index with one key part and a view with three logical keys, you could enter:

```
Correct usage: GET (view-name) USING SM* 1234 999
```

Retrieving multiple views

You may want to use more than one view in a program. For example, assume you want to perform a function which uses the three views listed below (the columns in each view are listed below the view name):

| CUSTOMER-ORDER-VIEW | CUSTOMER-VIEW      | PRODUCT-VIEW      |
|---------------------|--------------------|-------------------|
| Order Number        | Customer Number    | Part Number       |
| Customer Number     | Customer Name Part | Name              |
| Part Number         | Customer Address   | Part Cost         |
| Quantity Ordered    | Customer Telephone | Quantity in Stock |
| Part Cost           |                    |                   |
| Total Cost          |                    |                   |
| Ship Date           |                    |                   |

Perhaps you want to print the customer name and the part name ordered by a customer. The following steps show how to do this:

1. Place the required customer number and order number into the key columns CUSTOMER-NUMBER and ORDER-NUMBER:

COBOL

```
MOVE 12345 TO CUSTOMER-NUMBER.
MOVE 67890 TO ORDER-NUMBER.
```

FORTRAN

```
CUSTOMER_NUMBER = 12345
ORDER_NUMBER = 67890
```

BASIC

```
LET CUSTOMER_NO = 12345
LET ORDER_NO = 67890
```

- Retrieve the CUSTOMER-ORDER-VIEW (using the customer number and the order number as keys) to find the number of the part ordered:

**COBOL**

```
GET CUSTOMER-ORDER-VIEW USING CUSTOMER-NUMBER, ORDER-NUMBER.
```

**FORTRAN**

```
GET CUSTOMER_ORDER_VIEW USING CUSTOMER_NUMBER
+ORDER_NUMBER
```

**BASIC**

```
RDM GET CUSTOMER_ORDER_VIEW USING CUSTOMER_NO, &
ORDER_NO
```

- Let CUSTOMER-VIEW (customer number as a key) to find the customer's name.

**COBOL**

```
GET CUSTOMER-VIEW USING CUSTOMER-NUMBER.
```

**FORTRAN**

```
GET CUSTOMER_VIEW USING CUSTOMER_NUMBER
```

**BASIC**

```
RDM GET CUSTOMER_VIEW USING CUSTOMER_NUMBER
```

- Using part number as a key, retrieve PART-VIEW to find the part name.

**COBOL**

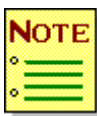
```
GET PART-VIEW USING PART-NUMBER
where PART-NUMBER comes from CUSTOMER-ORDER-VIEW
```

**FORTRAN**

```
GET PART_VIEW USING PART_NO
```

**BASIC**

```
RDM GET PART_VIEW USING PART_NO
```




---

Your DBA could do this more efficiently by combining the above statements into one view.

---

## Using MARK to save position of a row for later access

MARK causes the RDML processor to mark the current position of the view established by the previous GET, UPDATE, or INSERT.

### COBOL

```
MARK CUSTOMER-VIEW AT SAVE-LV.
```

### FORTRAN

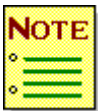
```
MARK PART_VIEW AT SAVE_LV
```

### BASIC

```
RDM MARK PART_VIEW AT SAVE_LV
```

The AT clause specifies where RDM should save the MARK on the view. You must define the column used. For example, define SAVE-LV (underscores for FORTRAN AND BASIC) in your COBOL program as a PICTURE X(4), FORTRAN program as a CHARACTER\*4, and BASIC program as a STRING = 4 column. You may use the AT clause in the GET to reread the row at the position set by the MARK even if you have performed other operations on this view.

## Using GET to control record holding



---

The considerations discussed in this section apply to all programs that access the PDM through RDM—that do not directly access the physical data.

---

Excessive record holding can cause a significant drop in performance. Some record holding is necessary to ensure data integrity. However, when other tasks attempt to access a held record, they must reset and try again until the record is released. These repeated resets increase processing time and impair performance.

In a multitasking environment, records held by another task are temporarily unavailable. RDM application programs use the RDML commands GET, GET FOR UPDATE, and UPDATE. GET retrieves the record but does not hold it; GET FOR UPDATE retrieves and holds the record; UPDATE updates the record. The duration of the record holding and the success of the update depend on how you combine these commands in a program. The following table shows three examples of retrieving and updating records and some considerations for each example:

| Purpose                               | Sample GET/UPDATE        | Considerations                                                                                                                                                                                                                                                                                                                                             |
|---------------------------------------|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Minimize record holding               | GET                      | Retrieves the record but does not hold it until later in the UPDATE. Other tasks CAN modify the record between the time your program accesses it and updates or deletes it.                                                                                                                                                                                |
|                                       | UPDATE                   | A data error occurs on the update if any columns have been changed and RDM returns an FSI of D, a VSI of C, and flags each changed column with an ASI of C. The changed columns that are flagged contain the original values, <i>not</i> the changed ones. Therefore, you can save the column values and retrieve the altered record to resolve conflicts. |
| Ensure an UPDATE succeeds             | GET FOR UPDATE<br>UPDATE | Retrieves and holds the record from the start, preventing any other program from updating the record. However, if many programs use the record, it may impair performance.                                                                                                                                                                                 |
| Ensure record stability during a READ | GET FOR UPDATE           | Holds the record without updating it. This ensures that other programs cannot update the record while you are reading it. However, if many programs use the record, it may impair performance.                                                                                                                                                             |

## Handling an unsuccessful RDM function

To handle unsuccessful RDM functions, you can supply an error paragraph or NOT FOUND clause. (For more information on error handlers, see “[Handling error conditions](#)” on page 141.) If you do not supply an error handler and you receive anything other than an \* or X FSI, RDM performs an automatic RESET and repositions you at the top of the view. (For a complete listing of FSI values and their meanings, see “[Validating data](#)” on page 115.) For example, if you perform a GET and then an UPDATE on a read-only view, the UPDATE fails and you are repositioned at the top of the view. The next unqualified GET returns the first row in the view.

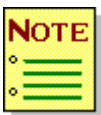
If you start your view processing with a keyed GET NEXT (default), a repeat of the same GET returns a NOT FOUND error. Because an error repositions you at the top of the view, another execution of the GET returns the correct row.

A data error will not lose position after an UPDATE or DELETE. (INSERT does not apply since the position of the view is determined by the INSERT.) If a data error is encountered in these cases, the program can request a GET SAME to look at the database row. When an error occurs, RDM does not alter the input row.

## Modifying rows

You can modify a row in three ways:

- ◆ Update the data already in the row (UPDATE)
- ◆ Delete the row (DELETE)
- ◆ Insert a new row (INSERT)



---

The DBA allows some views to modify the database, while other views are read-only.

---

When you modify rows, issue a COMMIT after each logical transaction (which might involve more than one change). COMMIT makes modifications permanent. For more information on using COMMIT, see “[COMMIT](#)” on page 167.

## Using UPDATE

Use UPDATE to modify the contents of columns. Before performing UPDATE, you use GET to access the view. For example:

### COBOL

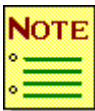
```
GET PART-VIEW USING PART-NAME
.
.
.
UPDATE PART-VIEW.
```

### FORTRAN

```
GET PART_VIEW USING PART_NAME
.
.
.
UPDATE PART_VIEW
```

### BASIC

```
RDM GET PART_VIEW USING PART_NAME
.
.
.
RDM UPDATE PART_VIEW
```



---

You cannot use UPDATE to modify a view key. The replacement of a view key is not allowed by the RDML processor since the view key locates the row to be replaced. To change a view key, first DELETE the old row, then INSERT a new one.

---

## Using DELETE

Use DELETE to remove a row from the database. Before performing DELETE, use GET to access the view.

This example deletes the one occurrence of PART-VIEW obtained based on the value of PART\_NAME:

### COBOL

```
GET PART-VIEW USING PART-NAME
DELETE PART-VIEW.
```

### FORTRAN

```
GET PART_VIEW USING PART_NAME
DELETE PART_VIEW
```

### BASIC

```
RDM GET PART_VIEW USING PART_NAME
RDM DELETE PART_VIEW
```

If a required column is not included in a user view, then a DELETE statement may delete more than one row.

The DELETE ALL starts at the beginning of the subset of rows with the specified keys and deletes each occurrence until a NOT FOUND condition exists. This example deletes all rows with the key value specified:

### COBOL

```
GET SAMPLE-VIEW USING KEY1
DELETE ALL SAMPLE-VIEW.
```

### FORTRAN

```
GET CUSTOMER_ORDER_VIEW USING CUSTOMER_NAME
DELETE ALL CUSTOMER_ORDER_VIEW
```

### BASIC

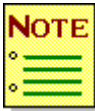
```
RDM GET CUSTOMER_ORDER_VIEW USING CUSTOMER_NAME
RDM DELETE ALL CUSTOMER_ORDER_VIEW
```



When you delete rows, consider the following constraints: To delete an entity from the database means to remove an object (a product). This differs from removing a relationship. If an employee transfers from one department to another, you do not remove the department. You remove the relationship between the employee and the first department.

Typically, a relationship delete can occur at any time. However, you cannot delete an entity if it is related to any other object. Therefore, to remove the employee record, you would first remove the relationship to the department as well as to any other objects.

You may delete an entity and all its relationships from the database. For example, if a customer cancels all outstanding orders and wants to be removed from your files, you first delete all relationships, and then delete the customer. You can do this in one RDML statement by coding `DELETE ALL` in the application program if your DBA allows such an operation on the view.



---

Use `DELETE ALL` with *extreme caution*.

---

## Using INSERT

Use INSERT to add a new user row to the database. The following example shows this:

### COBOL

```
INSERT CUSTOMER-VIEW.
```

### FORTRAN

```
INSERT CUSTOMER_VIEW
```

### BASIC

```
RDM INSERT CUSTOMER_VIEW
```

INSERT does not update any columns. For example, a view might contain both customer and account data. If you insert new account information into the view, then none of the existing customer information is updated, even if it has been changed in your data area. You must code an explicit update after the insert to update any columns present in the view before the insert.

**Controlling the placement of the row.** When inserting a user row in nonuniquely keyed rows, you can use NEXT, FIRST, LAST, or PRIOR to control the placement of the new row. You cannot determine the location if your DBA has already defined an order for the view. For example, INSERT NEXT ACCOUNT-DATA instructs SUPRA Server to insert the new row after the current one (the last row accessed).

If your DBA uniquely keyed the view, order is already determined. If the keys you are inserting already have values, RDM raises the DUP KEY condition and performs the action you specified on the DUP KEY phrase. The following sample INSERT shows this:

### COBOL

```
INSERT CUSTOMER-VIEW
 DUP KEY GO TO ALREADY-THERE.
```

### FORTRAN

```
INSERT CUSTOMER_VIEW
 DUP KEY
 GO TO 100
END IF
```

### BASIC

```
RDM INSERT CUSTOMER_VIEW
 DUP KEY
 GO TO 100
END IF
```

**Adding entities and relationships.** You can always add a new entity (a customer) assuming you have space on the database and the authority to INSERT. Typically, you cannot add a new relationship until all the entities being related exist. You cannot add a relationship between an employee and a department until the department and employee entities have been added.

However, in one operation you may add an entity and a relationship. For example, you may add an employee and the first department assignment in a single INSERT request, if the DBA allows this operation.

**Inserting null values.** Insert a null value on INSERTS or UPDATES only. To insert a null value, change the appropriate column's ASI to N prior to an RDML INSERT or UPDATE. The following statement inserts the null value into the column CUST-NAME in the view CUSTOMER:

### COBOL

```
MOVE "N" TO ASI-CUST-NAME
.
.
.
INSERT CUSTOMER
```

### FORTRAN

```
ASI_CUST_NAME= 'N'
.
.
.
INSERT CUSTOMER
```

### BASIC

```
CUSTOMER::ASI_CUST_NAME = "N"
.
.
.
INSERT CUSTOMER
```

## Controlling database recovery

Use COMMIT and RESET to control database recovery.

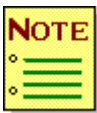
- ◆ COMMIT makes the changes to the database (INSERT, DELETE, UPDATE) permanent if the database has Task Level Recovery.
- ◆ RESET instructs SUPRA Server to perform the standard error recovery procedure for the previous RDML requests—to undo all database changes made by this task since the most recent COMMIT. If you do not supply an error handler (see “[Handling error conditions](#)” on page 141), a RESET request is automatically issued when a major error occurs.
- ◆ Sign-on and sign-off imply a COMMIT.

If the database contains RMS files, VMS RMS Recovery Unit Journaling may be enabled. Refer to the *[SUPRA Server PDM Database Administration Guide \(UNIX & VMS\)](#)*, P25-2260, for a description of how to enable Recovery Unit Journaling for physical files. Define the logical name CSI\_RMS\_RU\_ON to be TRUE before invoking DBAID. This allows RDM to log the transactions to the RMS files in a journal file that you can use to update or reset the RMS files if needed.

You can define CSI\_RMS\_RU\_ON TRUE in the group logical name table. In doing this you do not need to repeat the definition unless the machine on which you are working goes down. Alternatively, define CSI\_RMS\_RU\_ON TRUE before invoking DBAID as follows:

```
$DEFINE CSI_RMS_RU_ON TRUE
$RUN CSVDBAID
```

This ensures that records in RMS data sets are rolled back to the last successful COMMIT point if a system or application failure occurs. This matches Task Level Recovery for PDM data sets.



---

RMS Recovery Unit Journaling will not work across a network. RMS files marked for Recovery Unit Journaling are inaccessible from a remote node running RDM applications.

---

## Handling error conditions

Include an error handler in your program to indicate how RDM should handle errors. If you do not provide one, RDM generates a default error handler whenever a view request fails. There is a different error handler for each view and an error handler for those functions that do not involve a specific view.

### Understanding COBOL error handlers

To name an error handler, combine ERROR-ON- with the view name. For example:

```
ERROR-ON-PROD.
```

The following example illustrates a simple COBOL error handler:

```
PROD-TRAN.
 GET PROD USING TRAN-PROD
 NOT FOUND PERFORM ERROR-ON-PROD.
 .
 .
 .
ERROR-ON-PROD.
 DISPLAY "*** ERROR **" ULT-MESSAGE.
 DISPLAY " THIS JOB IS NOW TERMINATED ".
 RESET.
 SIGN-OFF.
 STOP RUN.
```

If you do not include the ERROR-ON-PROD paragraph, RDM generates the following default error paragraph for the view PROD:

```
ERROR-ON-PROD.
 RESET.
 SIGN-OFF.
 STOP RUN.
```

If you do not include a RESET before the SIGN-OFF, RDM performs an automatic COMMIT which might leave the database inconsistent. If you are coding a GET and the row you are retrieving might not exist, you should use the NOT FOUND clause. If you do not include it, the *error-handler* is entered if the row does not exist. You may also add phrases (such as DUP, ELSE, NOT FOUND, etc.) to your basic program statements to handle common exception conditions in your paragraph.

If an error occurs on a SIGN-ON, SIGN-OFF, COMMIT, RESET, RELEASE or FORGET, RDM performs ERROR-ON-ULT-CONTROL. Like other error handlers, you can either explicitly code an error handler or let RDM generate a default error handler which includes a RESET and STOP RUN.

## Loop prevention

If an RDM statement fails, RDM returns an FSI value other than : and your program branches to an error handler routine. For example, if the initial SIGN-ON failed and the RESET or SIGN-OFF fails, the following will loop.

```
ERROR-ON-ULT-CONTROL.
 DISPLAY "*** ERROR ***" ULT-MESSAGE.
 RESET.
 SIGN-OFF.
 STOP RUN.
```

You can include a test in the error handler to determine whether it has already been invoked, thereby avoiding the loop. For example:

```
01 ERROR-FOUND PIC X VALUE "N".
.
.
.
ERROR-ON-ULT-CONTROL.
 DISPLAY "*** ERROR ***" ULT-MESSAGE.
 IF ERROR-FOUND = "Y"
 STOP RUN.
 MOVE "Y" TO ERROR-FOUND.
 RESET.
 SIGN-OFF.
 STOP RUN.
```

You could also use the following error handler if a program terminates without doing a SIGN-OFF (RESET and SIGN-OFF are done automatically):

```
ERROR-ON-ULT-CONTROL.
 DISPLAY "*** ERROR ***" ULT-MESSAGE.
 STOP RUN.
```

A data error will not lose position after an UPDATE or DELETE. (INSERT does not apply since the position of the view is determined by the INSERT.) If a data error is encountered in these cases, the program can request a GET SAME to look at the database row. When an error occurs, the Relational Data Manager will not alter the input row.

## Understanding FORTRAN and BASIC error handlers

The following five considerations deal with the FORTRAN and BASIC error-handlers:

1. Each INCLUDE statement may have an associated ON ERROR statement, which introduces the error handler associated with this view or ULT-CONTROL. For example:

| <b>FORTRAN</b>    | <b>BASIC</b>          |
|-------------------|-----------------------|
| INCLUDE PART-VIEW | RDM INCLUDE PART-VIEW |
| ON ERROR          | RDM ON ERROR          |
| GO TO 999         | GO TO 999             |
| END ERROR-HANDLER | END ERROR HANDLER     |

The preprocessor expands this simple error handler in-line after any ACCESS statement which refers to PART-VIEW, instead of the default error-handler. RDM executes the error handler if any error occurs in the call to the RDM processor.

Any RDML statement that does not use any particular view (COMMIT, SIGN-OFF) will use the error handler associated with ULT-CONTROL (either the default error handler or the one specified after INCLUDE ULT-CONTROL). Each new subroutine which includes a view must specify an error handler, or SUPRA Server will use the default error handler.

2. The default error handler consists of a call to a FORTRAN subroutine, CSVDError, or a BASIC subroutine, CSVBError. The source of these subroutines is supplied in the SUPRA executable directory; you may modify or replace it. The supplied code displays the contents of ULT-CONTROL and performs a RESET and SIGN-OFF. You must link CSVDError to your FORTRAN RDML program and/or CSVBError to your BASIC RDML program.



The following is a listing of CSVDERROR:

## FORTRAN

```

1 1 SUBROUTINE CSVDERROR(PASSED_ULT_CONTROL)
 * This is the supplied default error-handling routine for
 * the CINCOM FORTRAN PREPROCESSOR. The passed copy of
 * ULT_CONTROL is used to construct the best error
messages
 * possible, given that the view itself cannot be passed
to
 * this general error routine. The routine then signs off
 * from RDM and stops the program. For ease of use, this
 * routine, or one similar to it, should be placed in an
 * object library accessible to the programmers using the
 * FORTRAN preprocessor.
2 ULTRA C INCLUDE ULT_CONTROL
3 ULTRA C ON ERROR
4 ULTRA C CONTINUE
5 ULTRA C END ERROR-HANDLER
6 8 CHARACTER*(ULT_CONTROL_LEN) PASSED_ULT_CONTROL
7 9 ULT_CONTROL=PASSED_ULT_CONTROL
8 10 TYPE
 100,ULT_FSI,ULT_VSI,ULT_OBJECT_NAME,ULT_OPERATION,
 1 ULT_MESSAGE
9 11 100 FORMAT(' ULT_FSI=',A1,'. ULT_VSI=',A1,'.
 OBJECT=' ,A30/
 1 '. OPERATION=' ,A6,'. MESSAGE=' ,A40,'.')
10 ULTRA C RESET
11 ULTRA C SIGN-OFF
12 27 STOP
13 28 END

```

The following is a listing of CSVBERROR:

## BASIC

```
100
SUB CSVBERROR(ULT_CONTROL_REC PASSED_ULT_CONTROL)
! This is the supplied default error-handling routine for the
! CINCOM BASIC PREPROCESSOR. The passed copy of ULT_CONTROL
! is used to construct the best error messages possible,
! given that the view itself cannot be passed to this general
! error-routine. The routine then signs off from RDM and
! stops the program. For ease of use, this routine, or one
! similar to it, should be placed in an object library
! accessible to the programmers using the BASIC preprocessor.
ULTRA ! RDM INCLUDE ULT_CONTROL
ULTRA ! RDM ON ERROR
ULTRA ! GOTO ERROR2
ULTRA ! END ERROR-HANDLER
PRINT ' ULT_FSI='; PASSED_ULT_CONTROL::ULT_FSI;
PRINT ' . ULT_VSI='; PASSED_ULT_CONTROL::ULT_VSI;
PRINT ' . OBJECT='; PASSED_ULT_CONTROL::ULT_OBJECT_NAME;
PRINT ' . OPERATION='; PASSED_ULT_CONTROL::ULT_OPERATION;
PRINT ' . MESSAGE='; PASSED_ULT_CONTROL::ULT_MESSAGE
ULTRA ! RDM RESET
ULTRA ! RDM SIGNOFF
ERROR2:STOP
END SUB
```

### 3. Handling DUP KEY and NOT FOUND statements:

- If you specify DUP KEY after an INSERT or the NOT FOUND after a GET statement, the error handler for the view is called only if the FSI status code is neither N nor \*. See “[Validating data](#)” on page 115 for a complete listing of FSI values.
- If the FSI is N, the statements after the DUP KEY or NOT FOUND are executed until a matching ELSE or END IF is found.
- If a matching ELSE is found, the statements after it are executed only if the FSI value is \*.

The following examples illustrate the use of the NOT FOUND and DUP KEY statements:

#### **FORTRAN**

```

GET PART_VIEW USING PART_NAME
NOT FOUND
 WRITE (*,*) 'PART', PART_NAME, 'NOT FOUND'
 GO TO 170
END IF
INSERT CUSTOMER_ORDER_VIEW
DUP KEY
 WRITE (*,*) 'THE ORDER ALREADY EXISTS'
 GO TO 180
ELSE
 WRITE (*,*) 'ORDER SUCCESSFULLY ADDED'
END IF

```

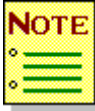
## BASIC

```
RDM GET PART_VIEW USING PART_NAME
NOT FOUND
 PRINT "PART" , PART_NAME, "NOT FOUND"
 GO TO 170
END IF
RDM INSERT CUSTOMER_ORDER_VIEW
DUP KEY
 PRINT "THE ORDER ALREADY EXISTS"
 GO TO 180
ELSE
 PRINT "ORDER SUCCESSFULLY ADDED"
END IF
```

4. A data error will not lose position after an UPDATE or DELETE. (INSERT does not apply since the position of the view is determined by the INSERT.) If a data error is encountered in these cases, the program can request a GET SAME to look at the database record. When an error occurs, RDM will not alter the input record.
5. The generated code for access statements, for example, GET, UPDATE, and so on, is the same as that for COBOL, except with FORTRAN or BASIC syntax. If the ACCESS statement does not use a view (SIGN-ON), then RDM uses the error handler for ULT\_CONTROL. The test for an error is the list of statements specified after the ON ERROR statement of the INCLUDE for the particular view being used. If no error clause is specified, then the procedure CSVDERROR (FORTRAN) or CSVBERROR (BASIC) is called.

## Implementing and executing an RDM program

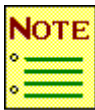
Before you run your RDML program, perform the following four steps (for a sample program in COBOL, FORTRAN, and BASIC, see the [Appendix](#) on page 375):



You must have access to the SUPRA Directory database, SUPRAD, to precompile your RDM program. You cannot precompile your program if the logical name CSI\_NODIRECTORY is set to TRUE.

1. Run the RDM preprocessor (Cincom supplies a command file that you can use which is located in the directory SUPRA\_COMS). When you run the preprocessor, it generates a source file and a file containing preprocessor errors (see the following table). Symbols for the preprocessors are defined in SUPRA\_COMS:SUPRA\_SYMBOL.COM.

| Language/Cincom-supplied command file    | Command     | Source file generated | Error file generated |
|------------------------------------------|-------------|-----------------------|----------------------|
| COBOL/<br>SUPRA_COMS:<br>RUNCOBOL.COM    | \$RUNCOBOL  | <i>filename.COB</i>   | <i>filename.COL</i>  |
| FORTRAN/<br>SUPRA_COMS:<br>RUNFORTRA.COM | \$RUNFORTRA | <i>filename.FOR</i>   | <i>filename.FOL</i>  |
| BASIC/<br>SUPRA_COMS:<br>RUNBASIC.COM    | \$RUNBASIC  | <i>filename.BAS</i>   | <i>filename.BAL</i>  |



The Cincom-supplied command file is set up to display a message on the terminal if an error occurs during preprocessing. To handle errors or warnings during preprocessing, you can set default actions by modifying this file. See the command file for suggestions on setting these default actions.

```
$RUNCOBOL
Enter COBOL Program Name and Extensions: INVSYSTEM.COP
Enter Database Name: INVOIC
Enter COBOL format (Term/ANSI)-[CR] = TERM:
```

```
$RUNFORTRA
Enter FORTRAN Program Name and Extension: INVSYSTEM.FOP
Enter Database Name: INVOIC
```

```
$RUNBASIC
Enter BASIC Program Name and Extension: INVSYSTEM.BAP
Enter Database Name: INVOIC
```

In the preceding figure, the Program Name and Extension is the name of the file containing your RDML statements before they are preprocessed. You must specify the file extension if it is not .COP (COBOL), .FOP (FORTRAN), or .BAP (BASIC). Database Name is the name of the database that all the views use.

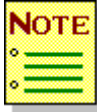
You could enter these specifiers on one line. For example:

**COBOL:** \$RUNCOBOL INVSYSTEM.COP INVOIC TERM

2. Run the VAX compiler. This generates an object file (*filename*.OBJ) and a listing containing any language syntax errors (*filename*.LIS) if specified. For example:

```
$COBOL /LIST/CROSS_REFERENCE INVSYSTEM.COB
$FORTRAN /LIST/CROSS_REFERENCE INVSYSTEM.FOR
$BASIC /LIST/CROSS_REFERENCE INVSYSTEM.BAS
```

3. Link your RDML program. This links your program with the run-time RDM system (RDML processor) and generates an executable file you can run.



CSVERROR.FOP and CSVBERROR.BAP must be precompiled and compiled prior to linking your RDML program. This task must only be done once. These files are delivered in the SUPRA\_EXE directory. If your DBA has placed a procedure CSVERROR (FORTRAN) or CSVBERROR (BASIC) in a system library, then you will not need to specify it on the link command line.

To link your program, use the LINK command:

### COBOL

```
$@SUPRA_COMS:CSVLINK list-of-module-names
Destination Directory:
```

### FORTRAN

```
$@SUPRA_COMS:CSVLINK list-of-module-names,CSVERROR
Destination Directory:
```

### BASIC

```
$@SUPRA_COMS:CSVLINK list-of-module-names,CSVBERROR
Destination Directory
```

4. Define the database your program uses if it's not already defined; then run your program as follows:

### COBOL

```
$DEFINE /USER CSI_SCHEMA database-name
$RUN program
```

### FORTRAN

```
$DEFINE /USER CSI_SCHEMA database-name
$RUN program
```

### BASIC

```
$DEFINE /USER CSI_SCHEMA database-name
$RUN program
```

The screen illustrations at the end of this section show an example of the whole process using the following program name:

## COBOL

INVSYSYSTEM.COP

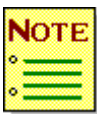
## FORTRAN

INVSYSYSTEM.FOP

## BASIC

INVSYSYSTEM.BAP

and INVOIC as the database name.



If you do not enter a directory in the Destination Directory field, the output will be placed in your current directory.

```
$RUNCOBOL
Enter COBOL Program Name and Extension: INVSYSYSTEM.COP
Enter Database Name: INVOIC
Enter COBOL format (term/ANSI) - [CR]=Term:
$COBOL/ANSI/LIST/CROSS_REFERENCE INVSYSYSTEM.COB
$@SUPRA_COMS:CSVLINK INVSYSYSTEM
Destination Directory:
$DEFINE/USER CSI_SCHEMA INVOIC
$RUN INVSYSYSTEM
```

```
$RUNFORTRA
Enter FORTRAN Program Name and Extension: INVSYSYSTEM.FOP
Enter Database Name: INVOIC
$FORTRAN/LIST/CROSS_REFERENCE INVSYSYSTEM.FOR
$@SUPRA_COMS:CSVLINK INVSYSYSTEM,CSVDERORR
Destination Directory:
$@SUPRA_COMS:RUNCSV INVSYSYSTEM
Enter Database Name: INVOIC
```

```
$RUNBASIC
Enter BASIC Program Name and Extension: INVSYSYSTEM.BAP
Enter Database Name: INVOIC
$BASIC/LIST/CROSS_REFERENCE INVSYSYSTEM.BAS
$@SUPRA_COMS:CSVLINK INVSYSYSTEM,CSVBERROR
Destination Directory:
$@SUPRA_COMS:RUNCSV INVSYSYSTEM
Enter Database Name: INVOIC
```

For sample COBOL, FORTRAN, and BASIC programs, see the [Appendix](#) on page 375.



# 5

## Coding RDM program statements (VMS)

RDM statements are divided into two groups: program data statements and program logic statements. For quick reference, this chapter provides the program statement names in the top, outside corner of each page.

---

### Coding program data statements

The Data Division of a COBOL program and the Declaration Statements of a FORTRAN or BASIC program describe the format and characteristics of data in an application program. If you are using COBOL, code the statements in either the Working-Storage Section or the Linkage Section of your program.

#### **INCLUDE *view-data***

Use INCLUDE to indicate the views your program needs.

---

#### **INCLUDE *view-data*: COBOL**

---

In COBOL you also use INCLUDE to indicate where (in the Data Division) you should code your views.

---

***level-number* [ *user-view-name*] INCLUDE *view-name* [(*data-item-list*)]**

---

---

**level-number**

|                    |                                                   |
|--------------------|---------------------------------------------------|
| <b>Description</b> | <i>Required.</i> A COBOL group item level number. |
| <b>Options</b>     | 01–29                                             |

---

**user-view-name**

|                    |                                                                               |
|--------------------|-------------------------------------------------------------------------------|
| <b>Description</b> | <i>Optional.</i> Assigns a valid view name to the user view for this program. |
|--------------------|-------------------------------------------------------------------------------|

---

**view-name**

|                    |                                                                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Indicates the name of the valid view you want to use. Your DBA or other authorized person must define the view on the Directory. |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------|

---

**data-item-list**

|                    |                                                                                                                                                                                                                                   |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Optional.</i> Allows you to use a subset of the columns in the view, indicating which ones you want to use.                                                                                                                    |
| <b>Format</b>      | <i>(data-item-name1,data-item-name2...data-item-namen)</i> A list of columns in the specified view. Separate the columns by commas and enclose them in parentheses. Spread the list over several continuation lines if necessary. |

**Considerations**

- ◆ If you specify no column list, all columns from the view are included.
- ◆ Required columns must have a valid, non-null value when inserting a row into a view. Therefore, if you do not include all REQUIRED columns into a user view, your program will not be able to perform INSERTS into the view.
- ◆ If you reorder logical keys in a view, you may inadvertently force RDM to read many rows from the view to satisfy a GET because no physical value is specified. The DBA has defined the key order on the Directory to maximize performance.

## General considerations

- ◆ Code an `INCLUDE` in the Data Division of your COBOL program to identify the view you want to use.
- ◆ Code an `INCLUDE` in the Linkage Section of your COBOL subroutines when rows are being passed as parameters.
- ◆ Use column lists to enhance the performance of your application. If your user view excludes all columns from a physical row, then RDM does not need to access that physical row to provide your user-view row.
- ◆ Each user view acts independently of all other user views included, even those from the same view. For example, if two user views are based on the same view, a `GET` on the first user view retrieves data defined in the first user view. You use another `GET` to retrieve the data defined for the second user view. In fact, the data requirements for both user views are obtained from the first row in the view.

**Examples**      The following examples generate data definitions in the Working-Storage Section:

**Input**

```
01 CUSTOMER INCLUDE CUST (CUST-NO,NAME,CITY).
```

**Output**

```
01 LUV-CUSTOMER.
10 CUSTOMER.
 20 CUST-NO PIC X(006).
 20 NAME PIC X(020).
 20 CITY PIC X(015).
10 ASI-CUSTOMER.
 20 ASI-CUST-NO PIC X.
 20 ASI-NAME PIC X.
 20 ASI-CITY PIC X.
```

**Input**

```
01 CONTACT INCLUDE PROD.
```

**Output**

```
01 LUV-CONTACT.
10 CONTACT.
 20 PROD-NO PIC X(006).
 20 PROD-DESC PIC X(040).
 20 PROD-RENT PIC S9(07)V9(02) USAGE COMP.
 20 PROD-MAINT PIC S9(07)V9(02) USAGE COMP.
 20 PROD-PURCH PIC S9(07)V9(02) USAGE COMP.
10 ASI-CONTACT.
 20 ASI-PROD-NO PIC X.
 20 ASI-PROD-DESC PIC X.
 20 ASI-PROD-RENT PIC X.
 20 ASI-PROD-MAINT PIC X.
 20 ASI-PROD-PURCH PIC X.
```

---

**INCLUDE *view-data*: FORTRAN and BASIC**

---

**FORTRAN**

---

**INCLUDE [COMMON] [*user-view-name*=]  
    *view-name* [(*data-item-list*) ]**

**[ON ERROR  
    *error-handler***

**END ERROR-HANDLER]**

---

**BASIC**

---

**RDM INCLUDE [COMMON] [*user-view-name*=]  
    *view-name* [(*data-item-list*) ]**

**[RDM ON ERROR  
    *error-handler***

**END ERROR-HANDLER]**

---

---

## COMMON

**Description**     *Optional.* Declares a common area having the name of the user view containing the record data area.

---

### *user-view-name=*

**Description**     *Optional.* Names the user view your program uses.

**Format**            Must be part of a valid view. Place the equal sign after the user view name.

### **Considerations**

- ◆ The user view name must use the same syntax as DBAID.
  - ◆ You should always assign a *user-view-name* to the *view-name* in the INCLUDE so that if a view name changes on the Directory, you only have to modify the *user-view-name* in the INCLUDE statement (not all places it occurs in your program).
- 

### *view-name*

**Description**     *Required.* Names the valid view (stored on the Directory) you want to use.

**(data-item-list)**

**Description**     *Optional.* Data items you want to use from the view.

**Format**            ([*user-data-item-name*=]*data-item-name1*,*data-item-name2*,...) A list of data items in the specified view. The data items are separated by commas and enclosed by parentheses. The list may be spread over several continuation lines if necessary.

**Considerations**

- ◆ If you specify no data item list, all data items from the view are included.
- ◆ Required data items must have a non-null value when inserting a record into a view. Therefore, if you do not include all REQUIRED data items into a user view, your program will not be able to perform INSERTS into the view.
- ◆ If you reorder logical keys in a view, you may inadvertently force RDM to read many records to satisfy a GET because your DBA did not specify physical values. (The DBA has defined the key order on the Directory to maximize performance.)
- ◆ You may rename each data item specified in the data item list if two views reference the same data name without specifying aliases.

[*new-data-item-name1*=] *data-item-name1*,

[*new-data-item-name2*=] *data-item-name2*,

[*new-data-item-name3*=] *data-item-name3*,.

---

## **ON ERROR (FORTRAN)**

### **RDM ON ERROR (BASIC)**

**Description** *Optional.* Introduces the error handler for this view.

**Consideration** Required if you code an error handler.

---

### ***error-handler***

**Description** *Optional.* A sequence of FORTRAN or BASIC statements with no labels and with no RDML statements.

**Consideration** These statements are expanded in-line. Therefore, if the view is used many times, it should not be too long.

---

## **END ERROR-HANDLER**

**Description** *Required* if you specify ON ERROR. Terminates the error handler for this view.

**Format** This is a single FORTRAN or BASIC RDML statement.



## General considerations

- ◆ Code an INCLUDE as a declaration of your program to identify the view you want to use.
- ◆ Code an INCLUDE as a declaration in your subroutines when a user record is being passed as a parameter.
- ◆ Using data item lists can enhance the performance of your application. If your user view excludes all data items from a physical record, then RDM does not need to access that physical record to provide your user view record.
- ◆ Each user view acts independently of all other user views included, even those from the same view. For example, if two user views are based on the same view, a GET on the first user view retrieves data defined in that user view. Use another GET to retrieve the data defined for the second user view. In fact, the data requirements for both user views are obtained from the first row in the view.
- ◆ Use underscores and hyphens as follows: All names in the INCLUDE statement may use either underscores (\_) or hyphens (-). However, these names will be expanded using the underscore instead of the hyphen. Use the underscore in the remainder of your program.
- ◆ (FORTRAN only) INCLUDE can be confused with both the FORTRAN text INCLUDE and with an assignment statement. If there is a quote following the word INCLUDE, the statement is assumed to be a FORTRAN text INCLUDE. If there is no space following the word INCLUDE, then the statement is assumed to be an assignment. Otherwise, the statement is assumed to be an RDML statement.
- ◆ You should always assign a *user-view-name* to the *view-name* in the INCLUDE. This minimizes the impact on your application of changing a view name as recorded on the Directory—you will only have to change the name in the INCLUDE, not every reference to the view name in your application program.

**Examples**

The following examples generate data definitions for the user view PARTS, consisting of the data items PART\_NAME and FABRICATION\_COST from the view V1. The first data item is renamed to V1\_PART. A data area is initialized (the names beginning ULT\$) to hold a description of the user view.

**FORTRAN**

```

C INCLUDE PARTS=V1(V1_PART=PART-NAME,FABRICATION-COST)
 CHARACTER*6 V1_PART
 INTEGER*4 FABRICATION_COST
 CHARACTER*1 ASI_V1_PART,ASI_FABRICATION_COST
 EQUIVALENCE (V1_PART,PARTS(1:6))
 BYTE ULB_FABRICATION_COST(4)
 EQUIVALENCE (ULB_FABRICATION_COST,FABRICATION_COST),(ULB
+ FABRICATION_COST,PARTS(7:10))
 EQUIVALENCE (ASI_V1_PART,PARTS(11:11))
 EQUIVALENCE (ASI_FABRICATION_COST,PARTS(12:12))
 INTEGER*4 PARTS_LEN
 PARAMETER(PARTS_LEN=12)
 CHARACTER*(PARTS_LEN)PARTS
 CHARACTER ULT$PARTS*30,ULT$V1_PART*1&,ULT$FABRICATION_COST*2$,
+ ULT_END_VIEW2*4
 DATA ULT$PARTS/'V1' /ULT$V1_PART
 +/'006C00PART-NAME,'/ULT$FABRICATION_COST
 +/'004B00FABRICATION-COST,'/ULT_END_VIEW2/'END.'/'
 CHARACTER*75 ULT$2
 EQUIVALENCE
 (ULT$2,ULT$PARTS),(ULT$2(31:47),ULT$V1_PART),(ULT$2(48
+ :71),ULT$FABRICATION_COST),(ULT$2(72:75),ULT_END_VIEW2

```

In FORTRAN, PARTS\_LEN may be used to declare parameters to a subroutine of the same length as the view PARTS. The intermediate variable ULB\_FABRICATION\_COST is declared only to avoid a FORTRAN or BASIC compiler warning.

**BASIC**

```

!RDM INCLUDE PARTS=V1(V1_PART=PART-NAME,FABRICATION-COST)
RECORD PARTS_REC
STRING V1_PART=6, LONG FABRICATION_COST
STRING ASI_V1_PART = 1, ASI_FABRICATION_COST = 1
END RECORD
DECLARE PARTS_REC PARTS
DECLARE STRING CONSTANT PARTS_ULT ="V1
"+
"006C00PART-NAME, "+"004B00FABRICATION-COST, "+"END."

```

## INCLUDE ULT-CONTROL

Use INCLUDE ULT-CONTROL to include the special view ULT-CONTROL in a program.

### INCLUDE ULT-CONTROL: COBOL

---

***level-number* INCLUDE ULT-CONTROL.**

---



---

#### ***level-number***

**Description**     *Required.* A COBOL group item level number.

**Options**            01–29

#### **General considerations**

- ◆ You must include ULT-CONTROL in each program which issues an RDML request. Subroutines which accept input views but perform no access themselves do not need this special view.
- ◆ Code an INCLUDE ULT-CONTROL in the Linkage Section of your application program if RDM is passing it from a calling module.

**Example**            The following adds ULT-CONTROL to your program.

```
01 INCLUDE ULT-CONTROL.
```

**Example output**   The following example illustrates the expansion of ULT-CONTROL:

```
01 ULT-CONTROL.
 10 ULT-OBJECT-NAME PIC X(30).
 10 ULT-OPERATION.
 20 ULT-ID PIC X(2).
 20 ULT-OPCODE PIC X.
 20 ULT-POSITION PIC X.
 20 ULT-MODE PIC X.
 20 ULT-KEYS PIC X.
 10 ULT-FSI PIC X.
 10 ULT-VSI PIC X.
 10 FILLER PIC X(2).
 10 ULT-MESSAGE PIC X(40).
 10 ULT-PASSWORD PIC X(8).
 10 ULT-OPTIONS PIC X(4).
 10 ULT-CONTEXT PIC X(4).
 10 ULT-LVCONTEXT PIC X(4).
```

---

**INCLUDE ULT-CONTROL: FORTRAN and BASIC**

---

**FORTRAN**

---

**INCLUDE [COMMON] ULT-CONTROL**

**[ON ERROR**

*error-handler*

**END ERROR-HANDLER]**

---

**BASIC**

---

**RDM INCLUDE [COMMON] ULT-CONTROL**

**[RDM ON ERROR**

*error-handler*

**END ERROR-HANDLER]**

---

---

**COMMON**

**Description**     *Optional.* Declares a common area named ULT-CONTROL.

---

**ON ERROR            (FORTRAN)****RDM ON ERROR    (BASIC)**

**Description**     *Optional.* Introduces the error handler for ULT-CONTROL.

**Consideration**   Required if you code an error handler.

---

***error-handler***

**Description**     *Optional.* A sequence of statements with no labels and with no RDML statements.

**Considerations**

- ◆ RDM uses this error handler after any RDML statement which does not specify a view.
  - ◆ The preprocessor expands these statements in-line. Therefore, the error handler should not be too long.
- 

**END ERROR-HANDLER**

**Description**     *Required* if you did specify ON ERROR. Terminates the error handler for this view.

**General considerations**

- ◆ You must include ULT-CONTROL in each program that issues an RDML request. Subroutines which accept input views but perform no access themselves do not need this special view.
- ◆ Code INCLUDE ULT-CONTROL in your application subroutine if it is being passed from a calling module.

**Example** To add the special view ULT-CONTROL to your FORTRAN or BASIC program, code the following statement:

### **FORTRAN**

```
INCLUDE ULT-CONTROL
```

### **BASIC**

```
RDM INCLUDE ULT-CONTROL
```

### **Example output FORTRAN**

```
C INCLUDE ULT-CONTROL
 CHARACTER ULT_OBJECT_NAME*30,ULT_OPERATION*6,ULT-
 FSI*1,ULT_VSI*1,
+ULT_FILLER*2,ULT_MESSAGE*40,ULT-PASSWORD*8,ULT_OPTIONS*4,
+ULT_CONTEXT*4,ULT_LVCONTEXT*4
 PARAMETER(ULT_CONTROL_LEN=100)
 CHARACTER*(ULT_CONTROL_LEN) ULT_CONTROL
 EQUIVALENCE (ULT_CONTROL(1:30),ULT_OBJECT_NAME(1:30)),
+ (ULT_CONTROL(31:36),ULT_OPERATION(1:6)),(ULT_CONTROL(37:37),
+ULT_FSI(1:1)),(ULT_CONTROL(38:38),ULT_VSI(1:1)),
+ (ULT_CONTROL(39:40),ULT_FILLER(1:2)),(ULT_CONTROL(41:80),
+ULT_MESSAGE(1:40)),(ULT_CONTROL(81:88),ULT_PASSWORD(1:8)),
+ (ULT_CONTROL(89:92),ULT_OPTIONS(1:4)),(ULT_CONTROL(93:96),
+ULT_CONTEXT(1:4)),(ULT_CONTROL(97:100),ULT_LVCONTEXT(1:4))
 CHARACTER*14 ULT_DATE_STAMP
 DATA ULT_DATE_STAMP/'19831114143849'/
```

### **BASIC**

```
! RDM INCLUDE ULT-CONTROL
RECORD ULT_CONTROL_REC
STRING ULT_OBJECT_NAME = 30, ULT_OPERATION = 6,
 ULT_FSI = 1,ULT_VSI = 1, ULT_FILLER = 2,ULTL_MESSAGE = 40,
 ULT_PASSWORD = 8, ULT_OPTIONS = 4, ULT_CONTEXT = 4,
 ULT_LVCONTEXT = 4
END RECORD
DECLARE ULT_CONTROL_REC ULT_CONTROL
EXTERNAL SUB CSVIPLV, CSVBERROR
DECLARE STRING CONSTANT ULT_DATE_STAMP = '19840830161953'
! RDM ON ERROR
! GOTO 999
! END ERROR-HANDLER
```

---

## Coding program logic statements

This section presents the program logic statements in alphabetical order. In COBOL they are referred to as Procedure Division statements.

### COMMIT

Use COMMIT to issue a COMMIT to the Physical Data Manager (PDM).

### COBOL

---

**COMMIT.**

---

### FORTRAN

---

**COMMIT**

---

### BASIC

---

**RDM COMMIT**

---

### General considerations

- ◆ COMMIT makes the changes to the database (INSERT, DELETE, UPDATE) permanent. RESET instructs SUPRA Server to perform the standard error recovery procedure for the previous RDML requests—to undo all database changes made by this task since the most recent COMMIT.
- ◆ Since the PDM issues a RESET and SIGN-OFF on any program that disappears without signing off—it abended or someone stopped it, always code a COMMIT in your program. This ensures that any changes made by your program will be permanent even if the program disappears.

**Examples** COMMIT identifies the recovery point of the task which just completed.

### COBOL

```
*COMMIT.
 MOVE "LVC---" TO ULT-OPERATION
 CALL "CSVIPLVS" USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL
 IF ULT-FSI NOT = "*"
 PERFORM ERROR-ON-ULT-CONTROL.
```

### FORTRAN

```
C COMMIT
 ULT_OPERATION='LVC---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+%REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDEROR (ULT_CONTROL)
END IF
```

### BASIC

```
! RDM COMMIT
ULT_CONTROL::ULT_OPERATION="LVC---"
CALL CSVIPLVS(ULT_CONTROL BY REF,ULT_CONTROL BY REF,ULT_CONTROL
 BY REF,
 ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI[]"*) THEN
 GOTO 999
END IF
```



## DELETE

Use DELETE to remove a row from the database.




---

Your DBA may not allow you to perform deletes.

---

### COBOL

---

**DELETE [ALL] *view-name*.**

---

### FORTRAN

---

**DELETE [ALL] *view-name***

---

### BASIC

---

**RDM DELETE [ALL] *view-name***

---



---

## ALL

**Description**     *Optional.* Deletes all rows depending on the logical keys specified by the preceding GET for this view.

**Consideration** This statement uses the parameters of the GET issued just before the DELETE.

---

## *view-name*

**Description**     *Required.* Specifies the valid, open view you want to delete.

## General considerations

- ◆ DELETE removes the entire current row.
- ◆ Use DELETE ALL with *extreme caution*.
- ◆ DELETE ALL deletes all rows having the same key values as specified by the USING clause in the most recent GET.
- ◆ DELETE ALL removes all rows in the view if the most recent GET had no qualifying USING clause.
- ◆ DELETE ALL does not commit, so the maximum number of rows which may be deleted at once depends on the size of the task log and on the maximum number of held rows allowed by the database. If you attempt to delete more than this number of rows, you receive an X status. Instead of doing a DELETE ALL, you may program a loop containing periodic commits.
- ◆ DELETES are not performed if data integrity is compromised—you may not delete a customer record until you delete all outstanding orders for that customer.
- ◆ When a DELETE request has failed with a status of X, perform a RESET to ensure database integrity. If you provide an error handling paragraph which does not RESET following an X status on DELETE, then it is possible that only part of the modification is done.
- ◆ (COBOL only) The COBOL preprocessor recognizes DELETE as an RDM request on two occasions: (1) when a valid view name follows DELETE, and (2) when ALL follows DELETE.
  1. If you use any other DELETE statement than the above two, a warning message states that the preprocessor skipped the statement and assumed it to be COBOL. The preprocessor gives the DELETE statement to the COBOL compiler as is, and does not generate code to handle the statement because it was not considered a legal RDM request.
  2. When DELETE is followed by ALL, the view specified is either valid or invalid. If it is a valid request, the COBOL preprocessor generates code to handle the request. If the request is invalid, the preprocessor generates an error message in the first listing and comments the DELETE statement out to the COBOL compiler.

**Examples****COBOL**

The following example deletes the one occurrence of SAMPLE-VIEW based on the value of KEY1.

```
GET SAMPLE-VIEW USING KEY1.
DELETE SAMPLE-VIEW.
```

This example deletes all user view rows dependent on the value in KEY1.

```
GET SAMPLE-VIEW USING KEY1.
DELETE ALL SAMPLE-VIEW.
```

This has the same effect as the following set of statements:

```
GET FIRST SAMPLE-VIEW USING KEY1.
MORE.
DELETE SAMPLE-VIEW.
GET NEXT SAMPLE-VIEW USING KEY1.
NOT FOUND GO TO DONE.
GO TO MORE.
DONE.
```

**FORTRAN and BASIC**

The following example deletes one occurrence of SAMPLE-VIEW:

**FORTRAN**

```
DELETE PART_COMP
```

**BASIC**

```
RDM DELETE TEST6E
```

**Example output FORTRAN**

```
ULT_OBJECT_NAME='PART_COMP'
 ULT_OPERATION='LVD---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP)
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF
```

**BASIC**

```
ULT_CONTROL::ULT_OBJECT_NAME="TEST6E"
 ULT_CONTROL::ULT_OPERATION='LVD---'
 CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6E BY REF,
 ULT_DATE_STAMP BY REF,TEST6E_ULT BY REF)
 IF (ULT_CONTROL::ULT_FSI[]" * ") THEN
 CALL CSVBERROR(ULT_CONTROL)
 END IF
```

This example deletes all user view rows dependent on the key values in the last GET statement:

**FORTRAN**

```
DELETE ALL PART_COMP
```

**BASIC**

```
RDM DELETE ALL TEST6E
```

**Example output FORTRAN**

```
ULT_OBJECT_NAME='PART_COMP'
 ULT_OPERATION='LVD--*'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP)
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF
```

**BASIC**

```
ULT_CONTROL::ULT_OBJECT_NAME="TEST6F"
 ULT_CONTROL::ULT_OPERATION="LVD--*"
 CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6F BY REF,
 ULT_DATE_STAMP BY REF,TEST6F_ULT BY REF)
 IF (ULT_CONTROL::ULT_FSI[]" * ") THEN
 CALL CSVBERROR(ULT_CONTROL)
 END IF
```

## FORGET

Use FORGET to remove the specified mark from the list of marks in use and free the storage allocated by a previously issued MARK.

### COBOL

---

**FORGET** *data-item*

**[NOT FOUND** *cobol-imperative-statement*

**[ELSE** *cobol-imperative-statement* **]].**

---

### FORTRAN

---

**FORGET** *data-item*

**[NOT FOUND** *sequence-of-fortran-statements*

**[ELSE** *sequence-of-fortran-statements* **]].**

---

### BASIC

---

**RDM FORGET** *data-item*

**[NOT FOUND** *sequence-of-basic-statements*

**[ELSE** *sequence-of-basic-statements* **]].**

---

## ***data-item***

**Description**     *Required.* Specifies what mark information RDM should forget.

**Format**             Must follow data item naming standards for the applicable language.

### **Considerations**

- ◆ Define the mark field data item with the following:
  - COBOL        PIC X(4).
  - FORTRAN    CHARACTER\*4
  - BASIC        STRING=4
- ◆ (COBOL only) Define the data item in the Data Division of the program.
- ◆ The data item must contain information returned by a previously issued MARK.

---

## **NOT FOUND *cobol-imperative-statement***

## **NOT FOUND *sequence-of-fortran-statements***

## **NOT FOUND *sequence-of-basic-statements***

**Description**     *Optional.* Indicates what RDM should do if the mark information cannot be released.

**Consideration** A mark value might not be found by RDM if one of the following conditions is true:

- ◆ A previous FORGET or RELEASE already removed the mark.
- ◆ A MARK has never been done on the data item.
- ◆ The marked data item was changed by your program.

**ELSE *cobol-imperative-statement*****ELSE *sequence-of-fortran-statements*****ELSE *sequence-of-basic-statements***

**Description**     *Optional.* Indicates what RDM should do if the mark information release is done.

**Consideration** Program control passes to the next statement if you do not specify an ELSE clause.

**General considerations**

- ◆ Once you issue a FORGET, RDM releases the indicated mark and you cannot regain it without issuing a new MARK.
- ◆ Release unwanted marks to maintain performance.

## GET

Use GET to retrieve a row from the indicated view.

### COBOL

---

|                                                                              |                                                                                                                                            |      |      |      |       |       |                          |                                                                                                                                                              |                                                                              |                       |
|------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|------|------|------|-------|-------|--------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------|-----------------------|
| GET                                                                          | <table border="0"><tr><td>NEXT</td></tr><tr><td>LAST</td></tr><tr><td>SAME</td></tr><tr><td>FIRST</td></tr><tr><td>PRIOR</td></tr></table> | NEXT | LAST | SAME | FIRST | PRIOR | view - name [FOR UPDATE] | <table border="0"><tr><td>USING <i>data - item</i><sub>1</sub> [... <i>data - item</i><sub>9</sub>]</td></tr><tr><td>AT <i>data - item</i></td></tr></table> | USING <i>data - item</i> <sub>1</sub> [... <i>data - item</i> <sub>9</sub> ] | AT <i>data - item</i> |
| NEXT                                                                         |                                                                                                                                            |      |      |      |       |       |                          |                                                                                                                                                              |                                                                              |                       |
| LAST                                                                         |                                                                                                                                            |      |      |      |       |       |                          |                                                                                                                                                              |                                                                              |                       |
| SAME                                                                         |                                                                                                                                            |      |      |      |       |       |                          |                                                                                                                                                              |                                                                              |                       |
| FIRST                                                                        |                                                                                                                                            |      |      |      |       |       |                          |                                                                                                                                                              |                                                                              |                       |
| PRIOR                                                                        |                                                                                                                                            |      |      |      |       |       |                          |                                                                                                                                                              |                                                                              |                       |
| USING <i>data - item</i> <sub>1</sub> [... <i>data - item</i> <sub>9</sub> ] |                                                                                                                                            |      |      |      |       |       |                          |                                                                                                                                                              |                                                                              |                       |
| AT <i>data - item</i>                                                        |                                                                                                                                            |      |      |      |       |       |                          |                                                                                                                                                              |                                                                              |                       |

[NOT FOUND *cobol-imperative-statement* ]

[ELSE *cobol-imperative-statement* ]

---



**NEXT**  
**LAST**  
**SAME**  
**FIRST**  
**PRIOR**

|                    |                                                            |                                                                                                                                                                            |
|--------------------|------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Optional.</i> Indicates the order of retrieval of rows. |                                                                                                                                                                            |
| <b>Default</b>     | NEXT                                                       |                                                                                                                                                                            |
| <b>Options</b>     | NEXT                                                       | Retrieves the next row with the specified keys. If you do not supply keys, RDM returns the next row in the view. If no current row exists, GET NEXT operates as GET FIRST. |
|                    | LAST                                                       | Retrieves the last row in the view with the specified keys. If no keys are given, RDM returns the last row.                                                                |
|                    | SAME                                                       | Retrieves the row just accessed if a current row exists. If no current row exists, RDM signals a NOT FOUND condition.                                                      |
|                    | FIRST                                                      | Retrieves the first row in the view with the specified keys. If no keys are given, RDM returns the first row.                                                              |
|                    | PRIOR                                                      | Retrieves the previous row with the specified keys. If no current row exists, GET PRIOR operates as GET LAST.                                                              |

### Considerations

- ◆ The order in which RDM returns rows from a view by the GET FIRST and GET NEXT defines a sequence on the rows in a view. GET LAST and GET PRIOR return the rows from the view in reverse order. If using indices to retrieve the view, the secondary key must be defined as REVERSE or BOTH for your program to use GET LAST and GET PRIOR. If the secondary key is not defined as such, then RDM returns an error when a program attempts GET LAST or a GET PRIOR. If you are not using indices to retrieve the view, you can do GET LAST or GET FIRST on a related data set, but not on a primary data set.
- ◆ If you do not check for a NOT FOUND, a series of GET NEXTs loops back to the first row and continues processing.

**view-name**

|                    |                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Names the view you want to use.                                            |
| <b>Format</b>      | Must be a valid view name or the <i>user-view-name</i> if you specified one on the INCLUDE. |

---

**FOR UPDATE**

|                    |                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Optional.</i> Allows you to lock out other users' modifications to the row you are retrieving. |
|--------------------|---------------------------------------------------------------------------------------------------|

**Considerations**

- ◆ GET...UPDATE retrieves the row, but does not hold it until later in the program when it does the UPDATE. Use this method to minimize row holding.
- ◆ GET FOR UPDATE...UPDATE retrieves and holds the row from the start. By the time the program updates the row, it will have held it for some time. This method ensures the success of the update by preventing any other program from updating the row. However, it can impair performance if used on rows which are accessed by many programs. Use this method if the update must succeed.
- ◆ GET FOR UPDATE... holds the row without updating it. This method ensures that other programs cannot update the row while you are reading it but will impact performance if used on rows which are accessed by many programs. Use this method to ensure that the row remains stable while you are reading it.

**USING *data-item*<sub>1</sub>[...*data-item*<sub>9</sub>]**

**Description**     *Optional.* Specifies the key values to use for accessing the view.

**Format**            The data items must be part of a valid view.

**Considerations**

- ◆ Do not use the USING phrase with GET SAME or an AT phrase.
- ◆ The number of keys specified in the GET statement must not exceed the number of keys in the data item list.
- ◆ Any trailing omitted keys are treated as generic keys. Using generic keys allows both direct access to a row and a sequential scan of many rows. RDM returns all occurrences of a particular unspecified data item, as long as the other keys are satisfied.
- ◆ The order of specified keys in the USING phrase must correspond to the order of key declarations (left to right) in your user view (see the INCLUDE statement, “**INCLUDE view-data**” on page 153). You cannot omit a key which occurs between two keys you want to specify.
- ◆ If a given key has only one row and you use that key with GET USING to access the row a second time, you receive a message indicating that there are no more occurrences with this logical key specification. In order to access the same row, use GET SAME.
- ◆ Up to nine data items may be used for the logical key.
- ◆ If a data item is a secondary key, you may omit characters from the right substituting the wildcard character \* or =. See “**Retrieving rows using partial keys**” on page 127. If using compound secondary keys, you may omit characters from the right of the last key only substituting one of the wildcard characters. You cannot use a wildcard character in the middle of a key data item.

---

### **AT data-item**

**Description**     *Optional.* Repositions a view based on the mark obtained by a previous MARK.

**Format**            Must be defined with a picture clause of PIC X(4).

#### **Considerations**

- ◆ The data item is a storage location which contains information generated by a previous MARK.
- ◆ You may not use the USING and AT phrases in the same GET.
- ◆ You may not specify the AT phrase in a statement using the FIRST, NEXT, PRIOR, LAST, or SAME positional qualifiers.

---

### **NOT FOUND *cobol-imperative-statement***

**Description**     *Optional.* Indicates what RDM should do if it does not find data.

#### **Considerations**

- ◆ Data might not be found for one or more of the following reasons:
  - No data is available for a keyed GET.
  - All the existing data is exhausted for a generic GET.
  - All the data available to the user view is exhausted for a non-keyed GET.
- ◆ If you do not check for a NOT FOUND, a series of GET NEXTs loops back to the first row and continues processing.
- ◆ If the row you are retrieving may not exist, you should include the NOT FOUND statement. If you do not include it, RDM enters its own error handler. For more information on error handlers, see [“Handling error conditions”](#) on page 141.

---

### **ELSE *cobol-imperative-statement***

**Description**     *Optional.* Indicates what RDM should do if it finds good data.

**Consideration** If you do not specify an ELSE statement, program control passes to the next statement.

**Examples**

The following statement retrieves the first row in the view PROD that matches the supplied key value. The column PROD-TRAN contains the key value used for retrieving the row:

```
GET PROD USING PROD-TRAN.
```

This statement retrieves the view using the key PROD-TRAN. The USING phrase indicates that RDM used a key to retrieve the view:

```
GET PROD FOR UPDATE USING PROD-TRAN.
```

This statement retrieves a view that a person marked and saved for later access:

```
GET PROD AT PROD-MARK.
```

Repeatedly issuing this request retrieves all PROD rows in the view:

```
GET PROD.
```

This request retrieves the row for update denying other processes update or delete access until a COMMIT or RESET is issued:

```
GET PROD FOR UPDATE.
```

The following statements retrieve rows in the specified order—NEXT, LAST, SAME, FIRST, and PRIOR:

```
GET NEXT PROD.
```

```
GET LAST PROD.
```

```
GET SAME PROD.
```

```
GET FIRST PROD.
```

```
GET PRIOR PROD.
```

The following statements illustrate the NOT FOUND and ELSE clauses:

```
GET PROD USING PROD-TRAN.
```

```
NOT FOUND GO TO PROD-NOT-FOUND.
```

```
ELSE DISPLAY PROD-TRAN "Retrieved Successfully".
```

These statements generate the following:

```
* GET PROD USING PROD-TRAN.
 MOVE "PROD " TO ULT-OBJECT-NAME
 MOVE PROD-TRAN
 TO PROD-NO
 OF PROD
 MOVE "LVG-R1" TO ULT-OPERATION
 CALL "CSVIPLVS" USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD
 IF ULT-FSI NOT = "*" and ULT-FSI NOT = "N"
 PERFORM ERROR-ON-PROD
 ELSE IF ULT-FSI = "N"
 GO TO PROD-NOT-FOUND
 ELSE DISPLAY PROD-TRAN "Retrieved successfully".
```

**FORTTRAN and BASIC**

---

GET 

|       |
|-------|
| NEXT  |
| LAST  |
| SAME  |
| FIRST |
| PRIOR |

*view - name* [FOR UPDATE] 

|                                   |
|-----------------------------------|
| AT <i>mark - variable</i>         |
| USING <i>key - expression,...</i> |

[NOT FOUND

*sequence-of-statements* ]

[ELSE

*sequence-of-statements* ]

END IF

---

|                                                       |                                                 |                                                                                                                                                                            |
|-------------------------------------------------------|-------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <div>NEXT<br/>LAST<br/>SAME<br/>FIRST<br/>PRIOR</div> |                                                 |                                                                                                                                                                            |
| Description                                           | Optional. Indicates the order of row retrieval. |                                                                                                                                                                            |
| Default                                               | NEXT                                            |                                                                                                                                                                            |
| Options                                               | NEXT                                            | Retrieves the next row with the specified keys. If you do not supply keys, RDM returns the next row in the view. If no current row exists, GET NEXT operates as GET FIRST. |
|                                                       | LAST                                            | Retrieves the last row in the view with the specified keys. If no keys are given, RDM returns the last row.                                                                |
|                                                       | SAME                                            | Retrieves the row just accessed if a current row exists. If no current row exists, RDM signals a NOT FOUND condition.                                                      |
|                                                       | FIRST                                           | Retrieves the first row in the view with the specified keys. If no keys are given, RDM returns the first row.                                                              |
|                                                       | PRIOR                                           | Retrieves the previous row with the specified keys. If no current row exists, GET PRIOR operates as GET LAST.                                                              |

Considerations

- ◆ The order in which RDM returns rows from a view by the GET FIRST and the GET NEXT functions defines a sequence on the rows in a view. GET LAST and GET PRIOR return the rows from the view in reverse order. If using indices to retrieve the view, the secondary key must be defined as REVERSE or BOTH for your program to use GET LAST and GET PRIOR. If the secondary key is not defined as such, then RDM returns an error when a program attempts GET LAST or a GET PRIOR. If you are not using indices to retrieve the view, you can do GET LAST or GET FIRST on a related data set, but not on a primary data set.
- ◆ If you do not check for a NOT FOUND, a series of GET NEXTs loops back to the first row and continues processing.



---

**view-name**

|                    |                                                                                                       |
|--------------------|-------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Names the view you want to use.                                                      |
| <b>Format</b>      | Must be a valid view name or the <i>user-view-name</i> if one was specified on the INCLUDE statement. |

---

**FOR UPDATE**

|                    |                                                                                                   |
|--------------------|---------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Optional.</i> Allows you to lock out other users' modifications to the row you are retrieving. |
|--------------------|---------------------------------------------------------------------------------------------------|

**Considerations**

- ◆ GET...UPDATE retrieves the row, but does not hold it until later in the program when it does the UPDATE. Use this method to minimize row holding.
  - ◆ GET FOR UPDATE...UPDATE retrieves and holds the row from the start. By the time the program updates the row, it will have held it for some time. This method ensures the success of the update by preventing any other program from updating the row. However, it can impair performance if used on rows which are accessed concurrently by many programs. Use this method if the update must succeed.
  - ◆ GET FOR UPDATE... holds the row without updating it. This method ensures that other programs cannot update the row while you are reading it but will impact performance if used on rows which are accessed by many programs. Use this method to ensure that the row remains stable while you are reading it.
- 

**AT mark-variable**

|                    |                                                                                    |                                  |
|--------------------|------------------------------------------------------------------------------------|----------------------------------|
| <b>Description</b> | <i>Optional.</i> Repositions a view based on the mark obtained by a previous MARK. |                                  |
| <b>Format</b>      | <b>FORTRAN</b>                                                                     | Must be a CHARACTER*4 data item. |
|                    | <b>BASIC</b>                                                                       | Must be a STRING=4 data item.    |

**Considerations**

- ◆ The variable is a storage location which contains information generated by a previous MARK.
- ◆ Do not use the USING and AT phrases in the same GET.
- ◆ Do not specify AT in a statement using the FIRST, NEXT, PRIOR, LAST, or SAME positional qualifiers.

## **USING *key-expression*,...**

**Description**      *Optional.* Specifies the key values to use for accessing the view.

**Format**            Any valid FORTRAN or BASIC expression which may be assigned to the required key data item.

### **Considerations**

- ◆ Do not use the USING phrase with a GET SAME or an AT.
- ◆ The number of keys specified in the GET must not be more than the number of keys in the data item list.
- ◆ Any trailing omitted keys are treated as generic keys. Using generic keys allows both direct access to a row and a sequential scan of many rows. RDM returns all occurrences of a particular unspecified data item as long as the other keys are satisfied.
- ◆ The order of specified keys in the USING phrase must correspond to the order of key declarations (left to right) in your user view (see the INCLUDE statement, “**INCLUDE view-data**” on page 153). You cannot omit a key which occurs between two keys you want to specify.
- ◆ If a given key has only one row and you use that key with GET USING to access the row a second time, you receive a message that there are no more occurrences with this particular logical key specification. In order to access this same row, use GET SAME.
- ◆ Up to nine data items may be used for the logical key.
- ◆ If you specify several values, separate them by commas.
- ◆ If using a secondary key, you may omit characters from the right substituting one of the wildcard characters \* or =. See “**Retrieving rows using partial keys**” on page 127. If you specify a secondary key consisting of more than one part, use the wildcard only on the rightmost part of the last data item in the key.

---

**NOT FOUND *sequence-of-statements***

**Description**     *Optional.* Indicates what RDM should do if it does not find data.

**Considerations**

- ◆ NOT FOUND expands into an IF statement. Therefore, you must place a matching END IF statement somewhere after it.
- ◆ NOT FOUND is only valid after a GET statement.
- ◆ Data might not be found for one or more of the following reasons:
  - No data is available for a keyed GET.
  - All the existing data is exhausted for a generic GET.
  - All the data available to the user view is exhausted for a non-keyed GET.
- ◆ If you do not check for a NOT FOUND, a series of GET NEXTs loops back to the first row and continues processing.
- ◆ If you are unsure whether the row you are retrieving exists, include the NOT FOUND statement. If you do not include it, RDM enters the error handler.

---

**ELSE *sequence-of-statements***

**Description**     *Optional.* Indicates what RDM should do if good data is found.

**Considerations**

- ◆ If you do not specify an ELSE statement, program control passes to the next statement.
- ◆ The ELSE statement is only valid after a NOT FOUND statement.

**END IF**

**Description**     *Required* after a NOT FOUND statement.

**Examples**        The following statement retrieves the first row in the view PART-COMP that matches the supplied key value. The column PART\_KEY contains the key value used for retrieving the row:

**FORTTRAN**

```
GET PART_COMP USING PART_KEY
```

**BASIC**

```
RDM GET PART_COMP USING PART_KEY
```

This statement retrieves a row marked and saved for later access:

**FORTTRAN**

```
GET PART_COMP AT PART_MARK
```

**BASIC**

```
RDM GET PART_COMP AT PART_MARK
```

Repeatedly issuing this request retrieves all PART\_COMP rows in the view:

**FORTTRAN**

```
GET PART_COMP
```

**BASIC**

```
RDM GET PART_COMP
```

This request retrieves the row for update, denying other processes update or delete access until COMMIT or RESET is issued.

### **FORTRAN**

```
GET PART_COMP FOR UPDATE
```

### **BASIC**

```
RDM GET PART_COMP FOR UPDATE
```

The following statements retrieve rows in the specified order—NEXT, LAST, SAME, FIRST, PRIOR:

### **FORTRAN**

```
GET NEXT PART_COMP
GET LAST PART_COMP
GET SAME PART_COMP
GET FIRST PART_COMP
GET PRIOR PART_COMP
```

### **BASIC**

```
RDM GET NEXT PART_COMP
RDM GET LAST PART_COMP
RDM GET SAME PART_COMP
RDM GET FIRST PART_COMP
RDM GET PRIOR PART_COMP
```

## INSERT

Use INSERT to insert a new row into the view.

### COBOL

---

```
INSERT [NEXT
 LAST
 FIRST
 PRIOR] view – name
```

[**DUP KEY** *cobol-imperative-statement* ]

[**ELSE** *cobol-imperative-statement* ].

---

### FORTRAN and BASIC

---

```
RDM INSERT [NEXT
 LAST
 FIRST
 PRIOR] view – name
```

[**DUP KEY** *sequence-of-statements*]

[**ELSE** *sequence-of-statements*]

**END IF**

---

**NEXT**  
**LAST**  
**FIRST**  
**PRIOR**

|                    |                                                                                                        |                                                                                                                                                                                           |
|--------------------|--------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Optional.</i> Specifies where RDM should insert the row relative to the current user view position. |                                                                                                                                                                                           |
| <b>Default</b>     | NEXT                                                                                                   |                                                                                                                                                                                           |
| <b>Options</b>     | NEXT                                                                                                   | Places the row after the current row, provided the keys are the same. If the keys are different, or if no current row exists, INSERT NEXT operates as INSERT LAST.                        |
|                    | LAST                                                                                                   | Places the row in the view so that a subsequent GET LAST using the same key values retrieves it.                                                                                          |
|                    | FIRST                                                                                                  | Places the row in the view so that subsequent GET FIRST using the same key values retrieves it.                                                                                           |
|                    | PRIOR                                                                                                  | Places the row in the view before the current row, provided the keys are the same. If the key values are different, or if there is no current row, INSERT PRIOR operates as INSERT FIRST. |

**Considerations**

- ◆ If the DBA specified ordering in the view definition or if the PDM does not allow program control of ordering, RDM ignores the specification on the INSERT.
- ◆ When an INSERT request has failed with a status of X, you must perform a RESET to ensure database integrity. If you provide an error handling paragraph which does not RESET following an X status on INSERT, then it is possible that RDM performed only part of the modification (this may result in logical corruption).

***view-name***

|                    |                                                                                  |
|--------------------|----------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Names the valid, open view in which you want the rows inserted. |
|--------------------|----------------------------------------------------------------------------------|

---

### **DUP KEY *cobol-imperative-statement***

### **DUP KEY *sequence-of-fortran-statements***

### **DUP KEY *sequence-of-basic-statements***

**Description**     *Optional.* Indicates what RDM should do if the row you are inserting is uniquely keyed and/or if the value of the keys you are inserting already exists in the database.

**Consideration** If you are using FORTRAN or BASIC, the DUP KEY statement expands into an IF statement. Therefore, you must place a matching END IF somewhere after it.

---

### **ELSE *cobol-imperative-statement***

### **ELSE *sequence-of-fortran-statements***

### **ELSE *sequence-of-basic-statements***

**Restriction**     Valid only after a DUP KEY statement.

**Description**     *Optional.* Indicates what RDM should do if the INSERT succeeds.

**Consideration** If you do not specify ELSE clause, program control passes to the next statement.

### **General consideration**

You must supply all keys and required columns in order for the INSERT to be successful.



**Examples**

The following examples show various ordering possibilities available for use with INSERT:

**COBOL**

```
INSERT NEXT PART-COMP.
INSERT LAST PART-COMP.
INSERT FIRST PART-COMP.
INSERT PRIOR PART-COMP.
```

**FORTRAN**

```
INSERT PART_COMP
INSERT NEXT PART_COMP
INSERT PRIOR PART_COMP
INSERT FIRST PART_COMP
INSERT LAST PART_COMP
```

**BASIC**

```
RDM INSERT PART_COMP
RDM INSERT NEXT PART_COMP
RDM INSERT PRIOR PART_COMP
RDM INSERT FIRST PART_COMP
RDM INSERT LAST PART_COMP
```

This example illustrates the code the RDM precompiler generates when you code DUP KEY and ELSE:

### COBOL

```

* INSERT PROD DUP KEY PERFORM DUP-PROD.
 MOVE "LVI---" TO ULT-OPERATION
 MOVE "PROD" TO ULT-OBJECT-NAME
 CALL "CSVIPLVS" USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD
 IF ULT-FSI NOT = "*" AND ULT-FSI NOT = "N"
 PERFORM ERROR-ON-PROD
 ELSE IF ULT-FSI = "N"
 PERFORM DUP-PROD.

```

### FORTRAN

```

C INSERT PART_COMP
 ULT_OBJECT_NAME='PART_COMP'
 ULT_OPERATION='LVI---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
C DUP KEY
 IF ((ULT_FSI.NE.'*') .AND. (ULT_FSI.NE.'N')) THEN
 CALL CSVDERROR(ULT_CONTROL)
 ELSE IF (ULT_FSI.EQ.'N') THEN
 GO TO 500
 ELSE
 TYPE *, 'Record inserted successfully.'
 END IF

```

### BASIC

```

! RDM INSERT FIRST TEST6G
ULT_CONTROL::ULT_OBJECT_NAME="TEST6G"
ULT_CONTROL::ULT_OPERATION="LVIFU-"
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6G BY REF,
ULT_DATE_STAMP BY REF,TEST6G_ULT BY REF)
! DUP KEY
IF ((ULT_CONTROL::ULT_FSI[]"*) AND (ULT_CONTROL::ULT_FSI[]"N"))
 THEN
 PRINT ULT_CONTROL::ULT_FSI
 GOTO 999
 END IF
IF (ULT_CONTROL::ULT_FSI="N") THEN
 GOTO 900
END IF

```

# MARK

Use MARK to record the current position of the view established by the most recent GET, UPDATE, or INSERT.

## COBOL

**MARK *view-name* AT *data-item*.**

## FORTRAN

**MARK *view-name* AT *mark-variable***

## BASIC

**RDM MARK *view-name* AT *mark-variable***

---

### *view-name*

**Description**     *Required.* Names the valid, open view you wish to mark.

---

### **AT *data-item* or AT *mark-variable***

**Description**     *Required.* Specifies where RDM should save the MARK information.

**Format**            Define the data item with PIC X(4) (COBOL), CHARACTER\*4 (FORTRAN) or STRING=4 (BASIC). data item.

### **Considerations**

- ◆ **COBOL.** You must define this data item in the Data Division of your program and initialize it to spaces before use.
- ◆ **FORTRAN.** Declare the mark variable as CHARACTER\*4 and initialize it to spaces.
- ◆ **BASIC.** Declare the mark variable as STRING=4 and initialize it to spaces.

## General considerations

- ◆ The AT clause in a GET (see “GET” on page 176, under the AT data-item parameter description) repositions the view at the position set by the MARK.
- ◆ You may create any number of MARKs for a view. A MARK might require several hundred bytes of internal memory if the view accesses several files with long keys.
- ◆ For efficient performance, use FORGET to remove marks that are no longer needed.

## Example COBOL

In this example RDM marks the current position of the user view PROD and saves it at PROD-MARK:

```
WORKING-STORAGE SECTION.
 .
 .
 .
01 PROD-MARK PIC X(4).
 .
 .
 .
PROCEDURE DIVISION.
 .
 .
 .
 MARK PROD AT PROD-MARK.
 .
 .
 .
 GET PROD AT PROD-MARK.
 .
 .
 .
```

## FORTRAN

In this example the current position of the user view PART\_COMP is marked and saved at PART\_MARK:

```

C MARK PART_COMP AT PART_MARK
 ULT_OBJECT_NAME= ' PART_COMP '
 ULT_OPERATION= ' LVM--- '
 CALL CSVIPLVS(%REF(ULT_CONTROL) , %REF(PART_COMP) ,
+ %REF(ULT_DATE_STAMP) , %REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDError(ULT_CONTROL)
 ELSE
 PART_MARK=ULT_CONTEXT
 END IF

```

## BASIC

In this example the current position of the user view TEST1 is marked and saved at TEST1\_MARK:

```

! RDM MARK TEST1 AT TEST1_MARK
ULT_CONTROL::ULT_OBJECT_NAME="TEST1"
ULT_CONTROL::ULT_CONTEXT=TEST1_MARK
ULT_CONTROL::ULT_OPERATION="LVM---"
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI[]"*)" THEN
CALL CSVBERROR(ULT_CONTROL)
ELSE
TEST1_MARK=ULT_CONTROL::ULT_CONTEXT
END IF

```

## RELEASE

Use RELEASE to close a specific view or all views that have been opened, and to release the occupied storage.

### COBOL

---

**RELEASE** [*view-name* ].

---

### FORTRAN

---

**RELEASE**

---

### BASIC

---

**RDM RELEASE**

---

---

#### *view-name* (COBOL only)

**Description**     *Optional.* Specifies the valid, open view RDM should release.

**Consideration** If you omit this parameter, RDM releases all of your opened views.

#### **General considerations**

- ◆ RELEASE is helpful when you are accessing multiple views. However, if used without a view name, it removes all marks (see “**MARK**” on page 195) and loses the current position in all views being used.
- ◆ If using COBOL, always follow a RELEASE with a period. If the COBOL preprocessor finds anything other than a period, it issues a warning message stating that it has skipped the statement and assumed it to be a COBOL statement.

**Examples**

These examples indicate that RDM should close all opened views.

**COBOL**

```

*
* RELEASE.
 MOVE "LVR---" TO ULT-OPERATION
 CALL "CSVIPVLS" USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL
 IF ULT-FSI NOT = "*"
 PERFORM ERROR-ON-ULT-CONTROL.

```

**FORTRAN**

```

C RELEASE
 ULT_OPERATION='LVR---'
 CALL CSVIPVLS(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

```

**BASIC**

```

! RDM RELEASE
ULT_CONTROL::ULT_OPERATION="LVR---"
CALL CSVIPVLS(ULT_CONTROL BY REF,ULT_CONTROL BY REF,ULT_CONTROL
BY REF,
ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI[]"*) THEN
 GOTO 999
END IF

```

## RESET

Use RESET to undo any UPDATE, DELETE, or INSERT statements issued since the most recent COMMIT.

### COBOL

---

**RESET.**

---

### FORTRAN

---

**RESET**

---

### BASIC

---

**RDM RESET**

---

### General considerations

- ◆ If you do not supply an error handler and an error occurs, RDM automatically issues a RESET request.
- ◆ If the database has Task Level Recovery, RESET backs out changes since the most recent COMMIT.



**Examples**

These examples indicate that a RESET is to be performed.

**COBOL**

```
*RESET.
 MOVE "LVA---" TO ULT-OPERATION
 CALL "CSVIPLV" USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL
 IF ULT-FSI NOT = "*"
 PERFORM ERROR-ON-ULT-CONTROL.
```

**FORTRAN**

```
C RESET
 ULT_OPERATION='LVA---'
 CALL CSVIPLV(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDError(ULT_CONTROL)
 END IF
```

**BASIC**

```
! RDM RESET
ULT_CONTROL::ULT_OPERATION="LVA---"
CALL CSVIPLV(ULT_CONTROL BY REF,ULT_CONTROL BY REF,ULT_CONTROL
BY REF,
ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI["*"]) THEN
 GOTO 999
END IF
```

## **SIGN-OFF**

Use SIGN-OFF to inform RDM that access to the database is no longer desired.

### **COBOL**

---

**SIGN-OFF.**

---

### **FORTRAN**

---

**SIGN-OFF**

---

### **BASIC**

---

**RDM SIGN-OFF**

---

### **General considerations**

- ◆ SIGN-OFF also releases all storage areas that were acquired to service RDML requests.
- ◆ Issue a SIGN-OFF at the end of every application program.
- ◆ SIGN-OFF also causes a COMMIT.

**Example**

In this example, the user signs off the system.

**COBOL**

```
*SIGN-OFF.
 MOVE "LVC---" TO ULT-OPERATION
 CALL "CSVIPLVS" USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL
 IF ULT-FSI NOT = "*"
 PERFORM ERROR-ON-ULT-CONTROL.
 MOVE 'LVF---' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL
 IF ULT-FSI NOT = "*"
 PERFORM ERROR-ON-ULT-CONTROL.
```

**FORTRAN**

```
C SIGN-OFF
 ULT_OPERATION='LVC---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF
 ULT_OPERATION='LVF---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF
 !
```

## BASIC

```
! RDM SIGN-OFF
ULT_CONTROL::ULT_OPERATION="LVC---"
CALL CSVIPVLS(ULT_CONTROL BY REF,ULT_CONTROL BY REF,ULT_CONTROL
 BY REF,
 ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI[]"*) THEN
 GOTO 999
END IF
ULT_CONTROL::ULT_OPERATION="LVF---"
CALL CSVIPVLS(ULT_CONTROL BY REF,ULT_CONTROL BY REF,ULT_CONTROL
 BY REF,
 ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI[]"*) THEN
 GOTO 900
END IF
```

# SIGN-ON

Use SIGN-ON to identify the user to RDM.

## COBOL

**SIGN-ON *user-name* [*password* ].**

## FORTRAN

**SIGN-ON *user-name* [,*password* ]**

## BASIC

**RDM SIGN-ON *user-name* [*password* ]**

---

### *user-name*

- Description**     *Required.* Indicates the name of the user.
- Format**             Must be assigned in the Directory.
- Consideration** (COBOL only) Specify the user name as a COBOL data item name and not a literal.

---

### *password*

- Description**     *Optional.* Indicates the user's password. You must supply the password if the user has been assigned a password in the Directory.
- Format**             Must be assigned in the Directory.
- Consideration** If you specify a password, it must be a COBOL data item name and not a literal.

## General considerations

- ◆ A SIGN-ON implicitly issues a release request and frees any previously allocated internal storage space.
- ◆ A CSV\_SETUP\_REALM routine is available to programs that access SUPRA Server databases through RDM. Use this routine to specify the mode in which RDM should open each data set. Data sets are usually opened in SHRE mode. You can specify either PRIV mode, preventing any other user from accessing the data set, or RDLY mode, preventing any modifications to the data set by this process.

You may call the CSV\_SETUP\_REALM routine before the program issues a SIGN-ON. Below is an example of the call, where ACCESS-CONTROL is the name of a data area within the program that contains the access-control specification. The access-control parameter must be passed by reference:

### COBOL

```
CALL "CSV_SETUP_REALM" USING ACCESS-CONTROL
```

### FORTRAN

```
CALL CSV_SETUP_REALM(%REF(ACCESS-CONTROL))
```

### BASIC

```
CALL CSV_SETUP_REALM(ACCESS-CONTROL BY REF)
```

- ◆ The format of the contents of the access-control parameter is the same as that of the access-control parameter of the SINON command, described in “**SINON**” on page 332. An example of the format for the REALM specification follows:

### COBOL

```
PROGNAMEDBNAMEACCESS..DSETMODE....[DSETMODE....]END.
```

### FORTRAN and BASIC

```
PROGNAMEDBNAMEACCESS..DSETMODE....[DSETMODE....]END.
```

|          |                                                           |
|----------|-----------------------------------------------------------|
| PROGNAME | The 8-character program name                              |
| DBNAME   | The 6-character name of the compiled database description |
| ACCESS   | The access-mode of the program; either RDONLY or UPDATE   |
| ..       | A 2-space filler, reserved                                |
| DSET     | A 4-character data set name                               |
| MODE     | The 4-character data set mode: SHRE, PRIV or RDLY         |
| ....     | Four spaces reserved for data set status codes            |
| END.     | Indicates the end of the parameter list                   |

**Examples      COBOL**

This example illustrates the use of the SIGN-ON command. Note that you code the MOVE and SIGN-ON statements and the remainder is code that the preprocessor generates.

```
 MOVE "JAMES-SMITH" TO USER-ID.
* SIGN-ON USER-ID.
 MOVE "LVS---" TO ULT-OPERATION
 MOVE USER-ID
 TO ULT-OBJECT-NAME
 MOVE SPACES TO ULT-PASSWORD
 CALL "CSVIPLVS" USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-ULT-CONTROL.
```

This example illustrates the use of the SIGN-ON command with a password.

```
*
* SIGN-ON USER-ID USER-PASSWORD.
 MOVE "LVS---" TO ULT-OPERATION
 MOVE USER-ID
 TO ULT-OBJECT-NAME
 MOVE USER-PASSWORD TO ULT-PASSWORD
 CALL "CSVIPLVS" USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL

 IF ULT-FSI NOT = ""
 PERFORM ERROR-ON-ULT-CONTROL.
```



## FORTRAN

In this example, 'JADOE' signs on to the system.

```

C SIGN-ON 'JADOE',PASSWORD
 ULT_OBJECT_NAME='JADOE'
 ULT_PASSWORD='PASSWORD'
 ULT_OPERATION='LVS---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

```

## BASIC

In this example, 'JADOE' signs on to the system.

```

! RDM SIGN-ON 'JADOE'
ULT_CONTROL::ULT_OBJECT_NAME='JADOE'
ULT_CONTROL::ULT_PASSWORD='
ULT_CONTROL::ULT_OPERATION='LVS--'
CALL CSVIPLVS(ULT_CONTROL BY REF,ULT_CONTROL BY REF,ULT_CONTROL
 BY REF,
 ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI[]'*') THEN
 GOTO 999
END IF

```

# UPDATE

Use UPDATE to update data item values in the database.



Your DBA may not allow you to perform an update.

## COBOL

UPDATE *view-name*.

## FORTRAN

UPDATE *view-name*

## BASIC

RDM UPDATE *view-name*

---

### *view-name*

**Description**     *Required.* Names the valid, open view you wish to update.

## General considerations

- ◆ Before performing an UPDATE, use GET to access the view.
- ◆ Use GET FOR UPDATE before using UPDATE when computing a new value for a row (incrementing a counter, etc.). If you are using UPDATE to place a value in a row, you do not need to issue GET FOR UPDATE which is not dependent on the values already present.
- ◆ You cannot update a view key. By altering the view key, you actually request repositioning of the view, not modification of the current row. To update a view key, you must first delete the old row, and then insert a new one.
- ◆ Follow an X failure status from an UPDATE request by a RESET to ensure database integrity. If you provide an error-handling paragraph which does not RESET following an X status on DELETE, then it is possible that RDM performed only part of the modification (this may result in logical corruption).

**Example      COBOL**

The statement UPDATE PROD indicates that the view PROD should be updated.

```
GET PROD USING ---
MOVE NEW-DATA TO PRODUCT-FIELD
UPDATE PROD.
.
.
.
```

**FORTRAN**

The statement UPDATE PART\_COMP indicates that the view PART\_COMP should be updated.

**Input**

```
UPDATE PART_COMP
```

**Output**

```
ULT_OBJECT_NAME=' PART_COMP '
ULT_OPERATION=' LVU--- '
CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+%REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
IF (ULT_FSI.NE.'*') THEN
CALL CSVDError(ULT_CONTROL)
END IF
```

**BASIC**

The statement RDM UPDATE TEST6D indicates that the view TEST6D should be updated:

**Input**

```
RDM UPDATE TEST6D
```

**Output**

```
ULT_CONTROL::ULT_OBJECT_NAME="TEST6D"
ULT_CONTROL::ULT_OPERATION="LVU---"
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6D BY REF,
ULT_DATESTAMP BY REF,TEST6D_ULT BY REF)
CALL CSVBError(ULT_CONTROL)
END IF
```

# 6

## Understanding Physical Data Manipulation Language (PDML)

To access and manipulate a database from an application program, you can use Physical Data Manipulation Language (PDML). It uses the Physical Data Manager (PDM) component of SUPRA Server, providing access to your physical databases. You use it to perform such functions as adding, reading and deleting primary and related records.

When using PDML, the first command your program must execute is a SINON. Then you can perform other activities: access, search, update data sets, and so on. If you update data sets, do periodic COMITs to flush buffers and free resources held by the task. The last command your program must execute (when your business function is complete) is a SINOFF. See the table under “[Table of PDML commands](#)” on page 229 for a complete description of each PDML command.

The DBA adds the data set descriptions into the Directory and assigns names to each field in a record. Your program uses those names in the data-list parameter of most PDML commands. A data list can name some or all of the fields in a record, in any order.

On a read, when the PDM returns control to the application (after servicing the command), the *data-area* parameter contains the contents of the data fields named in the data-list, in the same order.

The PDM locates each database record by its relative record number (RRN). Many PDML commands return an RRN as one of their outputs. Often, your program must pass this RRN to a subsequent DML command as an input parameter.

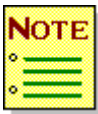
## Opening/closing data sets

Before any application can process a data set, the data set must be open. You can open a data set in two ways:

- ◆ Specify an entry in the realm segment of the SINON access-control parameter (see “**SINON**” on page 332).
- ◆ Allow the PDM to dynamically open the data set when you the first PDML command for that data set.

If you specify an entry in the realm segment of the SINON access-control parameter, you have either read only or update access to the data set. If you open the data set with read only access, the PDM returns an error status code to all DML commands issued from your program that attempt to update the data set.

If you allow the PDM to dynamically open the data set when you issue the first command, the PDM will open the data set for shared update access unless the data set is open for update access by another database. For valid combinations of modes see “**SINON**” on page 332.



---

You can not change a data set's mode while it is opened.

---

Regardless of how you opened the data set or in what mode, your program need not be concerned with closing the data set. When your program issues the SINOF PDML command the PDM logically closes the data sets which the application was using. A data set is only physically closed when the last task accessing it issues a SINOF DML command.

---

## Adding a primary record

Before your program can add a primary record, you must open the data set named in the data set parameter in UPDATE mode. You use ADD-M to add a record to a primary data set. The PDM locates a space for the new record using the information in the control key parameter. The new record is constructed using the information you place in the data list and data area parameters. Data items not named in the data list are filled with binary zeros in the added record. The PDM adds this record to the data set, and (in VMS only) performs maintenance to all secondary keys that are populated for this data set. See “[ADD-M](#)” on page 238 for more information on using ADD-M.

---

## Reading a primary record

When performing a read on a primary data set, you can navigate the data set in three ways:

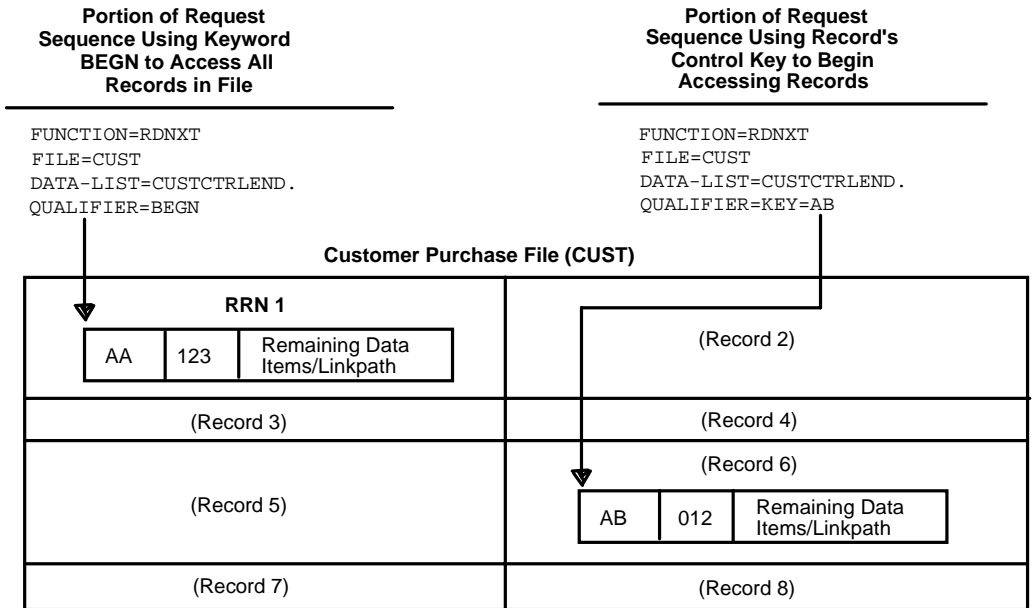
- ◆ A direct read reads one specific record, using the control key value of READM to perform a direct read of a primary data set. You can effect sequential processing by repeated reads, increasing or decreasing the specified key.
- ◆ A serial read means repeated reads for records in the sequence they are physically stored on the data set. For PDM data sets, this means the returned records are in random order. Use serial reads for large scale data set access or when the records to be retrieved are not known uniquely by key. RDNXT is available for serial reads on PDM data sets.
- ◆ A secondary key read processes the records in serial fashion, using the order determined by the secondary key definition. For Index data sets, the PDM reads records in key order. You can limit the reads to certain records by using masking information. Note that secondary keys may be defined for either primary or related data items.

Use a repeated READX to perform a secondary key serial read of a data set. Because all secondary keys are in either ascending or descending character sequence or both, you can retrieve the records using either a forward or reverse direction on the secondary key values. The first read must use the keyword BEGN in the qualifier field. To alter the direction of the read, specify the opposite direction keyword in options, and specify REBD in the qualifier; this starts reading in the opposite direction. You can start at the beginning or end of the index data set, or with a particular secondary key. See “**READX**” on page 314 for more information on using READX.

Use a repeated RDNXT command to perform a serial read of the data set. You can start at the beginning of a data set and continue through it, or start at a specified record and then continue. To start at the physical beginning, use the keyword BEGN in the qualifier parameter. To start processing with a specific record, use the control key (of the record where you want the search to begin) in the qualifier parameter. Repeat the RDNXT to continue to the end of data set. The PDM returns the data of each record and its location (RRN). See “**RDNXT**” on page 289 for more information on using RDNXT.



The following figure illustrates which record the PDM retrieves from a data set (CUST) when you code the RDNXT qualifier parameter as BEGN and when you use KEY=control key.



Data returned to your program when the qualifier is set to:

|                                      | BEGN | KEY=control-key |
|--------------------------------------|------|-----------------|
| Qualifier contents after first RDNXT | 1    | 6               |
| Data area contents after first RDNXT | AA   | AB              |

You may want to add (ADD-M) records to or delete (DEL-M) records from a primary data set while serially processing it with a RDNXT or sequentially processing it with READX. Use caution, because your application program may not have serial access to certain records after the PDM performs the delete and add logic. For example, the current record (retrieved serially) may have a synonym that has not been read. If the PDM deletes the current record, the PDM automatically reorganizes the data set and may physically move the synonym so it is unavailable for the next or a subsequent serial access. To avoid this situation, you should issue all ADD-Ms and DEL-Ms after serial processing is complete. This technique ensures that all records are available for program analysis. You can execute WRITM while processing a primary data set with RDNXT because WRITM does not reorganize synonym chains.

---

## Updating a primary record

Before you can update a primary record with WRITM, you should issue a read command. In a multitask operating mode or when task logging is active, you should issue a read (READM) with record holding to ensure that another task cannot hold the record you are using. The PDM uses the specified control key to locate the record to be updated. The PDM then moves the data items (as specified in the data list) from the data area to the corresponding sections in the record you want to update.

---

## Deleting a primary record

Use DEL-M to delete a record from a primary data set. The PDM deletes the record whose key is in the control key parameter from the data set identified by the data set parameter. If you are running your program in a multitask environment or if task logging is active, you should read (READM) the record with record holding before you attempt to delete it. You cannot delete the primary record if any related data set records are linked to it. See “**DEL-M**” on page 276 for more information on using DEL-M.

---

## Adding a related record

Before your program can add a related record, you must open the data set named in the data set parameter in UPDATE mode. You use ADDVC, ADDVB, or ADDVA to add a record to a related data set. The new record is constructed using the information you place in the data list and data area parameters. Data items not named in the data list are filled with binary zeros in the added record. The PDM adds this record to the data set, and performs structural maintenance to the linkpaths for the data set. In addition, the PDM performs maintenance to all secondary keys that are populated for this data set.

You can add new records to a chain of related records at various logical positions within the chain. A chain is a set of PDM related record connected to each other through the common linkpath field. The entire chain is connected to one primary file record through its linkpath field of the same name. If the related record has more than one linkpath, it is a member of more than one chain. You can add to the beginning, end, or at any logical position within the chain. When adding records to a related record chain, the PDM stores all records belonging to one chain as close together as possible. The following text and figures explain database navigation when you use an ADDVC, ADDVB, or ADDVA.

Use ADDVC (add chain) to add a related record at the logical end of the chain on the controlling linkpath. The PDM also adds the new record to the end of all other linkpaths defined for this record.

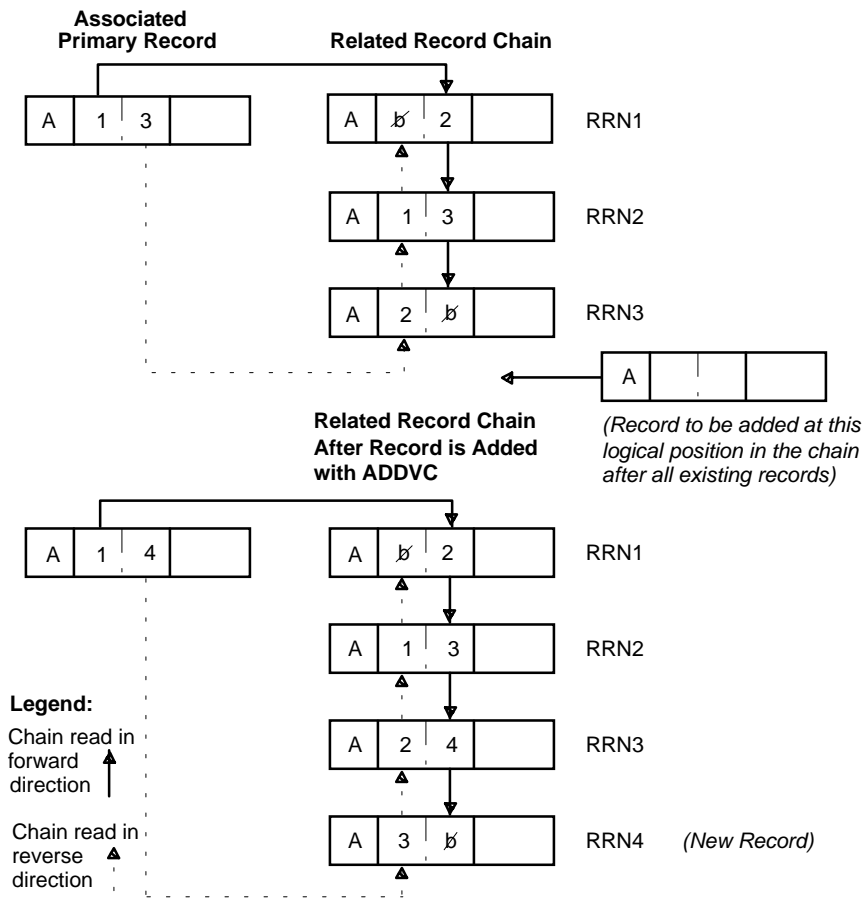


---

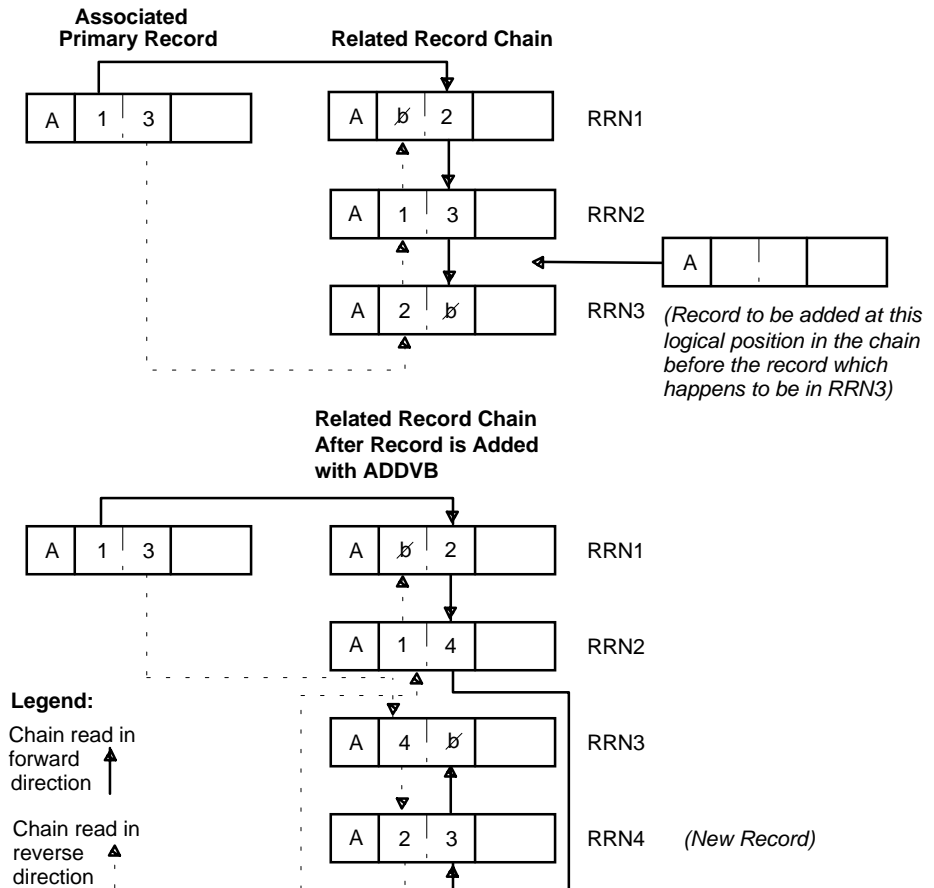
Cincom strongly recommends that all additions to a related data set use the same linkpath. This will help keep all records in one chain close together and thus give better performance. This should be the path of most frequent access.

---

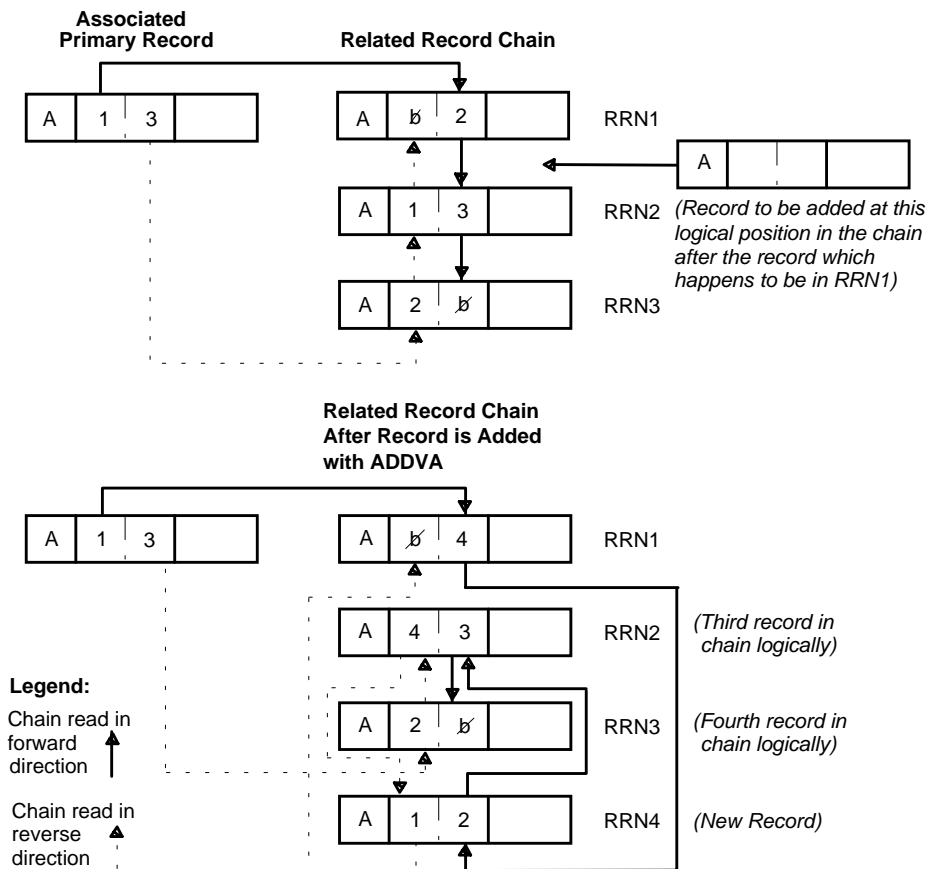
The following figure shows a primary record connected to a chain of three related records and what happens to the database's logical structure when you issue an ADDVC. Follow the arrows in this figure to see how the PDM navigates through a single chain of related records.



Use ADDVB (add before) to add a related record in front of another related record in a record chain. Before making this addition, you must know the RRN of the record before which you want to place the new record; otherwise, the PDM will not know where to place the new record. (Issue a read command to get the RRN of a record.) The PDM adds the new record logically in front of the record whose RRN you specify in the reference parameter. The new record is added to the end of all other linkpaths with which this record is associated. The following figure shows what happens to the database's logical structure when you issue an ADDVB. This figure also shows how the PDM navigates (following linkpaths) through a single chain of related records.



Use ADDVA to add (add after) a related record logically after another related record in a record chain. Before making this addition, you must know the RRN of the record after which you want to place the new record; otherwise, the PDM will not know where to place the new record. (Issue a read command to get the RRN of a record.) The PDM adds the new record logically after the record whose RRN you specify in the reference parameter. The new record is added to the *end* of all other associated linkpaths. The following figure shows what happens to the database's logical structure when you issue an ADDVA. This figure also shows how the PDM navigates through a single chain of related records. For more information on using ADDVA, ADDVB, and ADDVC, see “**ADDVA**” on page 242, “**ADDVB**” on page 249, and “**ADDVC**” on page 256.



---

## Reading a related record

When performing a read on a related data set, you can navigate the data set in four ways:

- ◆ A direct read reads one specific record. The PDM locates the record to be read by first using the control key parameter to locate the associated primary record. This primary record's linkpath points to the appropriate chain of related records. Then, using the RRN specified in the reference parameter, the PDM goes directly to the required related record.
- ◆ A serial read means repeated reads for records in the sequence they are physically stored on the data set (in RRN order).
- ◆ For related data set reads, the term sequential is similar to serial in that it indicates a type of read in which any number of records are read one after another. However, a sequential read processes the records in a logical sequence along linkpath chains.
- ◆ Secondary key reads are serial read processed in the order you specify on the READX. Secondary keys can be read in a forward or reverse direction starting at any point in the index data set.

Use READD (read direct) to perform a direct read of a related data set. READD directly reads a specific related record in a data set without reading any other related or primary record. For more information on using READD, see “READD” on page 294.

To perform strictly sequential reads, use a repeated READV or READR. Use READV to perform a forward sequential read of a linkpath set from its logical beginning to its logical end. Use the READR to perform a reverse sequential read of a linkpath set from its logical end to its logical beginning. For more information on using READV and READR, see “READV” on page 309 and “READR” on page 303.

Use repeated RDNXT command to perform a serial read of a specified related data set. For RDNXT, you can start the search at the beginning of a data set and continue through it, or start at a specified record and then continue. To start at the physical beginning, use the keyword BEGN in the qualifier parameter. To start processing with a specific record, use an RRN in the qualifier parameter. Repeat the RDNXT without changing the qualifier to continue to the end of data set. The PDM returns the data of each record and its location. See “[READM](#)” on page 300 for more information on using RDNXT.

In summary, when processing related data sets the order of data set navigation can be serial without regard to record chains, sequential without regard to physical order.

Use a repeated READX to perform a secondary key serial read. Because all secondary keys are in ascending or descending character sequence, the records can be retrieved using either a forward or reverse direction. The first read must use the keyword BEGN in the qualifier field. To alter the direction of the read, specify the opposite direction keyword in options, and specify REBD in the qualifier; this starts reading in the opposite direction. You can start at the beginning or end of the index data set, or with a particular secondary key. See “[READX](#)” on page 314 for more information on using READX.

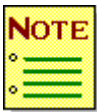


## Updating a related record

You can use WRITV or ADDVR to update a related record. When updating a related record, determine if the data set is coded or uncoded. If you are using a coded data list and the data set is coded, you must include the record code in the data list and data area. If these two codes do not match or if the code in the record does not match the code in the data area, an error status code is returned.

If the data set is uncoded, use WRITV. Also use WRITV for a coded data set if you are not changing a record code or control key. The PDM uses the RRN in the reference parameter to locate the related record. Before you can update a related record with WRITV, you must issue a read command to obtain the RRN. In a multitask operating mode or when task logging is active, you should issue a read command with record holding. When the PDM processes a WRITV, like the WRITM, the data items in the data area are moved to the corresponding location in the record you want to update.

Use ADDVR to update a related record when you want to change a record code or control key; otherwise use WRITV. The ADDVR command can remove a related record from one chain and link it to the end of another chain.

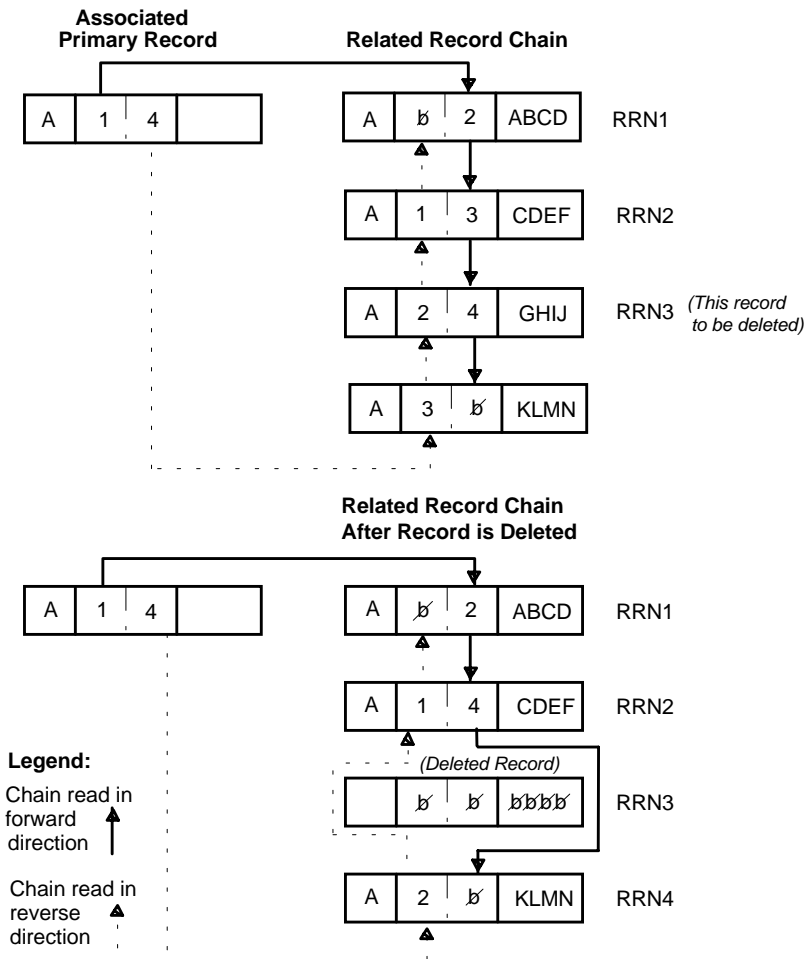


ADDVR is the *only* command you can use to change a record code or a control key.

ADDVR performs like the WRITV except when the update requires the addition or deletion of a linkpath in the record. If the record exists in both the old and new linkpaths and the key changes, the PDM logically relinks an existing related record into different chains without physically moving the record. The PDM examines every control key defined for the record to be processed by comparing the new data area to the PDM's I/O area. (The PDM's I/O area contains the existing record which is to be updated.) If a control key changes, the PDM removes the record from the old chain and logically adds the record to the end of the new chain; the new linkpath is determined by the control key specified in the data area. The PDM updates the old linkpath and new linkpaths to reflect the new links. See “**ADDVR**” on page 263 and “**WRITV**” on page 342 for more information on using ADDVR and WRITV.

## Deleting a related record

Use DELVD to delete a record from a related data set. If you are running your program in a multitask environment or if task logging is active, you should explicitly hold the record before you can delete it. (If you do not explicitly hold it first, you may receive a HELD status when you try to delete it.) The PDM removes the record whose RRN is in the reference parameter from the data set identified in the data set parameter. The PDM also removes the record from all associated linkpaths and fills the record with binary zeros so it is available for immediate reuse (see the following figure). After executing, the PDM updates the reference parameter with the RRN of the record immediately preceding the deleted record (in the record chain) identified by the linkpath parameter. See “**DELVD**” on page 278 for more information on using DELVD.



# 7

## Using PDML

Use PDML to access the PDM and manipulate data on your SUPRA Server physical databases.



---

**Warning:** Coding RDML and PDML commands in one program can cause database corruption.

---

The database access program, DATBAS, provides all access to the SUPRA Server databases. Physical DML (PDML) provides the format and values of the parameters for DATBAS. You can use any language conforming to the operating system calling standard to write your application program. The number of required parameters varies by command.

A typical PDML command is:

```
CALL "DATBAS" USING command-name, status, data-set, physical-key,
data-item-list, data-area, endp
```

where:

|                       |                                                                                                                          |
|-----------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>DATBAS</b>         | is the database access program in the PDM.                                                                               |
| <i>command-name</i>   | represents the 5-byte name of the DML function to be performed.                                                          |
| <i>status</i>         | represents a 4-byte area in your program to which the PDM returns the completion status of the requested function.       |
| <i>data-set</i>       | represents the data set you want to access.                                                                              |
| <i>physical-key</i>   | represents the physical key of the record you want to access.                                                            |
| <i>data-item-list</i> | represents a list of specific data items you want read from or written to the SUPRA Server database.                     |
| <i>data-area</i>      | represents the area in your program which contains, or will contain, the data items named in the <i>data-item-list</i> . |
| <i>endp</i>           | represents a special value which terminates the command parameter list, and indicates whether the record is to be held.  |

When the function call statement is completed, the PDM returns control to your application program. Your program should check the returned status code to determine the function result.

### Examples

- ◆ The same command in COBOL is:

```
CALL "DATBAS" USING UFUNC, USTAT, UDSET, CKEY,
CLIST, CAREA, UENDP
```

- ◆ The same command in FORTRAN is:

```
CALL DATBAS (UFUNC, USTAT, UDSET, CKEY, CLIST, CAREA, UENDP)
```

- ◆ (VMS) The same command in BASIC is:

```
CALL DATBAS (UFUNC, USTAT, UDSET, CKEY, CLIST, CAREA, UENDP)
```

- ◆ The same command in C is:

```
DATBAS (ufunc, ustat, udset, ckey, clist, carea, uendp);
```

---

## Table of PDML commands

The following tables provide a description and section reference for each PDML command. The figure following these tables specifies the parameters required for each command.

### Primary data set commands

| Command | Description                                                                | Section             |
|---------|----------------------------------------------------------------------------|---------------------|
| ADD-M   | Adds a record to a primary data set.                                       | "ADD-M" on page 238 |
| DEL-M   | Deletes a record from a primary data set.                                  | "DEL-M" on page 276 |
| READM   | Reads a record from a primary data set.                                    | "READM" on page 300 |
| WRITM   | Updates the indicated data items in a previously retrieved primary record. | "WRITM" on page 338 |

**Related data set commands**

| Command | Description                                                                             | Section                    |
|---------|-----------------------------------------------------------------------------------------|----------------------------|
| ADDVA   | Adds a related record to a list of records in the position after the specified record.  | <b>“ADDVA”</b> on page 242 |
| ADDVB   | Adds a related record to a list of records in the position before the specified record. | <b>“ADDVB”</b> on page 249 |
| ADDVC   | Adds a related record to the end of the list.                                           | <b>“ADDVC”</b> on page 256 |
| ADDVR   | Relinks an existing related record into different lists.                                | <b>“ADDVR”</b> on page 263 |
| DELVD   | Deletes the specified related record.                                                   | <b>“DELVD”</b> on page 278 |
| READD   | Retrieves a specified related record                                                    | <b>“READD”</b> on page 294 |
| READR   | Reads a related record along the reverse direction of a related record list.            | <b>“READR”</b> on page 303 |
| READV   | Reads a related record along the forward direction of a related record list.            | <b>“READV”</b> on page 309 |
| WRITV   | Updates the indicated data items in a previously retrieved related record.              | <b>“WRITV”</b> on page 342 |

**Serial processing command**

| Command | Description                                            | Section                    |
|---------|--------------------------------------------------------|----------------------------|
| RDNXT   | Reads primary or related records in physical sequence. | <b>“RDNXT”</b> on page 289 |
| READX   | Reads records in sequence using a secondary key.       | <b>“READX”</b> on page 314 |

## Special function commands

| Command | Description                                                                                                                                        | Section                        |
|---------|----------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------|
| COMIT   | Signals the completion of a logical unit of work, and physically writes all updates to the appropriate data sets.                                  | "COMIT" on page 273            |
| CNTRL   | (VMS only ) Holds single or multiple records.                                                                                                      | "CNTRL (VMS only)" on page 269 |
| MARKL   | Places user-specified data on the function log.                                                                                                    | "MARKL" on page 283            |
| OPCOM   | Passes requests to the PDM and receives output from the PDM.                                                                                       | "OPCOM" on page 285            |
| RESET   | Resets all changes made by this task in the current logical unit of work.                                                                          | "RESET" on page 324            |
| RQLOC   | Returns the home location of a primary record.                                                                                                     | "RQLOC" on page 326            |
| SINOF   | Disconnects the task from SUPRA Server.                                                                                                            | "SINOF" on page 328            |
| SINON   | Identifies the task to the PDM, specifies which database and data sets are required, and gives the access mode for the database and the data sets. | "SINON" on page 332            |

The following figure shows the parameters required for each PDML command. The parameters are positional; they must appear in the order indicated.

|                   | PARAMETERS |          |         |           |           |          |          |              |         |                |             |           |                |      |
|-------------------|------------|----------|---------|-----------|-----------|----------|----------|--------------|---------|----------------|-------------|-----------|----------------|------|
|                   | status     | data-set | options | qualifier | reference | linkpath | nodename | physical-key | command | data-item-list | area-length | data-area | access-control | endp |
| PRIMARY FUNCTIONS |            |          |         |           |           |          |          |              |         |                |             |           |                |      |
| ADD-M             | •          | •        |         |           |           |          |          | •            |         | •              |             | •         |                | •    |
| DEL-M             | •          | •        |         |           |           |          |          | •            |         | •              |             | •         |                | •    |
| READM             | •          | •        |         |           |           |          |          | •            |         | •              |             | •         |                | •    |
| WRITM             | •          | •        |         |           |           |          |          | •            |         | •              |             | •         |                | •    |
| RELATED FUNCTIONS |            |          |         |           |           |          |          |              |         |                |             |           |                |      |
| ADDVA             | •          | •        |         |           | •         | •        |          | •            |         | •              |             | •         |                | •    |
| ADDVB             | •          | •        |         |           | •         | •        |          | •            |         | •              |             | •         |                | •    |
| ADDVC             | •          | •        |         |           | •         | •        |          | •            |         | •              |             | •         |                | •    |
| ADDVR             | •          | •        |         |           | •         | •        |          | •            |         | •              |             | •         |                | •    |
| DELVD             | •          | •        |         |           | •         | •        |          | •            |         | •              |             | •         |                | •    |
| READD             | •          | •        |         |           | •         | •        |          | •            |         | •              |             | •         |                | •    |
| READR             | •          | •        |         |           | •         | •        |          | •            |         | •              |             | •         |                | •    |
| READV             | •          | •        |         |           | •         | •        |          | •            |         | •              |             | •         |                | •    |
| WRITV             | •          | •        |         |           | •         | •        |          | •            |         | •              |             | •         |                | •    |
| SERIAL FUNCTIONS  |            |          |         |           |           |          |          |              |         |                |             |           |                |      |
| RDNXT             | •          | •        |         | •         |           |          |          |              |         | •              |             | •         |                | •    |
| READX             | •          | •        | •       | •         |           |          |          |              |         | •              | •           | •         |                | •    |
| SPECIAL FUNCTIONS |            |          |         |           |           |          |          |              |         |                |             |           |                |      |
| CNTRL             | •          |          | •       |           |           |          |          |              |         | •              | •           | •         |                | •    |
| COMIT             | •          |          |         |           |           |          |          |              |         |                | •           | •         |                | •    |
| MARKL             | •          |          |         |           |           |          |          |              |         |                | •           | •         |                | •    |
| OPCOM             | •          |          |         | •         |           |          | •        |              | •       |                | •           | •         |                | •    |
| RESET             | •          |          |         |           |           |          |          |              |         |                | •           | •         |                | •    |
| RQLOC             | •          | •        |         |           |           |          |          | •            |         |                |             | •         |                | •    |
| SINON             | •          |          |         |           |           |          |          |              |         |                |             |           | •              | •    |
| SINOF             | •          |          |         |           |           |          |          |              |         |                |             |           | •              | •    |



## Data list parameter keywords

The data list tells the PDM what data items to place in the data area (for reads) or what is already in the data area (for writes and adds). The data list and data area must match, or results are unpredictable.

Normally you code the data list as the Directory physical field names of the data items you want to process. However, you can use some keywords in your data list, or as your entire data list, to perform special functions. Always use END. to end a data list. These keywords are:

**\*\*BIND\*\***, **\*FILL=nn**, **\*CODE=xx**, and **\*COMMON\***. The **\*COMMON\*** keyword retains compatibility for existing TOTAL or TIS applications.

---

### **\*\*BIND\*\***

Resolves the data list by constructing information in memory to eliminate a table lookup each time the data list is used. SUPRA automatically binds all data lists by using the entire data list as the key into the internal data list table. If used, **\*\*BIND\*\*** must be the first word in the data list. You cannot use commas between items in a data list. For example:

```
BINDdataitem1dataitem2...dataitemnEND.
```

On the first execution of the command with this list, the PDM changes **\*\*BIND\*\*** to **\*BNDxxxx**, where **xxxx** is an internal binary identifier used as a key into the internal data list table.

You can use this keyword for either primary or related files.

**\*FILL=nn**

Skips the specified number of bytes in the data area during data transfer. Multiple usage in the same data list is allowed. You can use this keyword for either primary or related files. You can use values of 00–99 for *nn*.

You initialize the data area to spaces or any values you want before using a data list with *\*FILL=nn* keyword(s). You might use *\*FILL=nn* to preformat a data area so that you can print directly from it after retrieving a record. For example, to generate a line of print that looks like:

```
CODE=02 KEY=00001 KEY=00002
```

You would first initialize the data area to:

```
CODE= KEY= KEY=
```

You would then code the data list as follows:

```
*FILL=05rrrrCODE*FILL=06rrrrKY01*FILL=06rrrrKY02END.
```

The PDM skips the first 5 bytes, which you have initialized to CODE=, returns the *rrrrCODE* value into the next 2 bytes, skips the next 6 bytes, and so on, resulting in the line you wanted.

When processing add or write commands, do not use *\*FILL=nn* unless there are also data items in the data list.

When read-ahead buffering is active, the behavior of *\*FILL=* can be slightly modified. The values passed in the data area are not necessarily the ones used to fill the data area of the returned records. When DATBAS requests multiple records from PDM, the data area containing the *\*FILL=* values is passed to PDM and is used to fill all of the records read. This works for most applications. However, if your application modifies the *\*FILL=* data on each DATBAS call, you will have to turn off the read-ahead feature. This can be done by defining the logical name CSI\_READAHEAD to NO. Refer to the *SUPRA Server PDM System Administration Guide (UNIX)*, P25-0132, for more details on defining logical names and the use of the CSI\_READAHEAD logical name.

---

**\*CODE=xx**

This keyword is for coded related files only, to process only certain record codes. For each occurrence, it identifies which record code the following subdata list refers to. Except for **\*\*BIND\*\***, this keyword must be the first item in the data list. It must not be the last item in the list; at least one data item must follow it.

On most read commands, the PDM merely skips any records with record codes not identified in the data list. However, if the read command is a READD, the PDM returns an error status if the code is not identified. On a write or add command, the PDM performs the request only if the record in the data area has one of the record codes identified in the data list. Otherwise the PDM returns an error status. Use **\*CODE=xx** as follows:

```
*CODE=xxdataitem1*CODE=xxdataitem1dataitem2END.
```

The following examples further describe the action the PDM takes on the various types of read, and on adds or updates. All examples refer to the following data list:

```
*CODE=01RECSElm1*CODE=02RECSElm1RECSkey1END.
```

1. When issuing a READD command (read direct), the PDM uses the RRN in the READD's reference parameter to locate the record to be read. Then, the PDM fills the data area with different information depending on the record code.

According to the example data list, if the RRN points to a record whose record code is 02, the data area contains:

```
xy00001
```

If the RRN points to a record whose record code is 01, the data area contains:

```
xy
```

However, if the READD RRN points to a record whose record code is 03, the PDM returns an error status code.

When using multiple record codes in a single element list, to know which record code has been returned, Cincom suggests that you include the RECSCODE element name in each coded sublist as follows:

```
*CODE=01RECSCODERECSelm1*CODE=02RECSCODERECSelm1RECSkey1END.
```

2. When performing a series of reads using RDNXT, READR, or READV for a coded file, using the above example data list, the PDM returns information as described below to the data area and status.

Records containing record codes other than those specified in the data list will be skipped when processing a RDNXT, READR, or READV. The PDM simply skips those records and processes the next record. You do not receive an error status as you would on a READD (see #1).

3. When performing an ADDVA, ADDVB, ADDVC, ADDVR or WRITV with the example coded data list, the PDM processes the request only if the data area record code is in the data list. The first 2 bytes of the data area determine which portion of the data list maps the remainder of data in the data area (\*CODE=01 or \*CODE=02). If another record code is in the data area, the PDM returns an error status.

---

**\*COMMON\***

This keyword is a compatibility form of \*CODE=xx, to make the data list a coded data list. Two formats are supported for existing TOTAL and TIS 1.x applications:

The following rules apply when coding a \*COMMON\* data list:

- ◆ Use the \*COMMON\* keyword for compatibility only. For new applications, use \*CODE=xx (or use the RDML language for the application).
- ◆ A data list should follow each \*CODE=xx, entry. If you do have data items common to more than one record code, name them in the \*COMMON\* base data list or in each \*CODE=xx sublist, as appropriate.
- ◆ Do not name the record code item in any coded data list unless you also name it as the first item in the base data list.
- ◆ When writing or adding a record (using the \*COMMON\* keyword), include the record code as the first data item in the base data list. This allows the PDM to determine which of the coded data lists maps the data area. If you omit the record code or include it somewhere other than at the beginning of the base data list, results are unpredictable.

# PDML commands

This section presents the PDML commands in alphabetical order. Each subsection presents a command with a general description followed by the command format, parameter descriptions, coding information, and operational considerations. All parameters shown within the command format are required.

## ADD-M

Use ADD-M (Add Primary) to add a new record to a primary data set. To determine the location of the new record on the data set, the Physical Data Manager (PDM) uses the relative address calculator on the unique physical key. The PDM then constructs the new record, using the required data items from the data area, and writes the record.

See “[The ADD-M command](#)” on page 361 for more information on using ADD-M successfully.

**ADD-M, status, data-set, physical-key, data-item-list, data-area, endp**

### status

**Description**      *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### Considerations

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the [SUPRA Server PDM Messages and Codes Reference Manual \(PDM/RDM Support for UNIX & VMS\)](#), P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

---

**data-set**

- Restriction**     *Required.* Specifies the name of the required data set.
- Format**            4-byte field
- Consideration**   The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter.

---

**physical-key**

- Description**     *Required.* Points to a field containing the key of the primary record to be added. The PDM uses this key to determine whether a record with the same physical key already exists.
- Format**            Variable-length as defined on the Directory
- Consideration**   During the command processing, if the *physical-key* parameter does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should use the *physical-key* field name in the data area for this parameter, rather than define a separate field.

---

**data-item-list**

- Description**     *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “[Data list parameter keywords](#)” on page 233.)
- Format**            Variable-length field in the following format:  
                      dataitem1dataitem2...dataitemnEND.

## Considerations

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Physical keys (CUSTCTRL)
  - Special data item processing keywords (\*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.
- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.
- ◆ The following is an example of a data item list:  
`CUSTCTRLCUSTADDRCUSTCREDEND.`

---

**UNIX**

Data items omitted from the list and data area are filled with either blanks or nulls, depending on the value of the binary-zero-key field in the database definition. If binary zero keys are allowed, the omitted fields are filled with spaces. If binary zero keys are not allowed, omitted fields are filled with binary zeros.

---



---

**data-area**

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

**Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

---

**endp**

**Description**     *Required.* Delimits the parameter list and indicates the record holding function.

**Format**            4-character field

**Options**            END.                    Holds the record after it is read

                        RLSE                    Delimits the parameter list (same effect as END.)

**General considerations**

- ◆ Data items you do *not* code in the data item list are set to binary zeros.
- ◆ The data item list holds names, and the data area holds a value for each of those names. Include the record's physical key name in the data item list and its value in the corresponding position of the data area.



Data items omitted from the list and data area are filled with either blanks or nulls, depending on the value of the binary-zero-key field in the database definition. If binary zero keys are allowed, the omitted fields are filled with spaces. If binary zero keys are not allowed, omitted fields are filled with binary zeros.

---

## ADDVA

Use ADDVA (Add Related After) to logically add the record in the data area after the record whose RRN is in the reference parameter. This logical addition is made only for the linkpath specified by the linkpath parameter—primary linkpath. For all other linkpaths defined for this record (secondary linkpaths), the addition is made to the end of each respective list.

---

**ADDVA, status, data-set, reference, linkpath, physical-key, data-item-list, data-area, endp**

---

---

### status

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

#### Considerations

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

---

### data-set

**Description**     *Required.* Specifies the name of the required data set.

**Format**            4-byte field

**Consideration** The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter

---

**reference**

|                      |                                                                                                                               |                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>   | <i>Required.</i> Points to a field identifying the RRN of the record after which the new record in this chain is to be added. |                                                                                                                                                                                                     |
| <b>Format</b>        | 4 alphanumeric characters or a 4-byte binary integer                                                                          |                                                                                                                                                                                                     |
| <b>Options</b>       | LKxx                                                                                                                          | The last 4 characters of the linkpath named by the linkpath parameter (where xx = the last 2 characters). The PDM adds the record to the end of the linkpath rather than after a particular record. |
|                      | rrrr                                                                                                                          | The RRN of the record after which the new record is to be added to the linkpath.                                                                                                                    |
| <b>Consideration</b> | After successful execution, this parameter contains the RRN of the record just added.                                         |                                                                                                                                                                                                     |

**linkpath**

**Description**     *Required.* Specifies the name of the linkpath as defined in the compiled database description. All related record functions use this parameter to indicate which related record list is being processed.

**Format**            8-byte field with the following format:

*ppppLKxx*

where:

*pppp*                represents the name of an associated primary data set.

LK                   is a literal (type in as shown).

*xx*                   represents the last 2 characters of the linkpath name as defined in the database description.

**Considerations**

- ◆ The primary data set contains the controlling primary record whose key is given in the primary key parameter. LKxx are the same characters used in the reference parameter to identify the beginning of the list. If you specify an invalid linkpath, the PDM returns MLNF, indicating that the linkpath is not defined in the compiled database description.
- ◆ There are two types of linkpaths: primary and secondary. A primary linkpath is the linkpath specified in add commands—ADDVC, ADDVA, ADDVB and ADDVR. Secondary linkpath refers to all other linkpaths defined for a particular record in the compiled database description.
- ◆ To choose the primary linkpath for a record, you need to know the average length of each record list and the frequency of access along each list. Generally, the primary linkpath should be either the longest record list or the most frequently accessed. When using PDML, *always* use the primary linkpath for ordering records.

---

***physical-key***

- Description**     *Required.* Points to a field containing the key of the record in the primary file named in the linkpath parameter.
- Format**           Variable-length as defined on the Directory
- Consideration**   During the command processing, if the *physical-key* parameter does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should use the *physical-key* field name in the data area for this parameter, rather than define a separate field.

---

***data-item-list***

- Description**     *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “[Data list parameter keywords](#)” on page 233.)
- Format**           Variable-length field in the following format:
- dataitem1dataitem2...dataitemnEND.*

## Considerations

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Record codes (ORDRCODE)
  - Special data item processing keywords (\*CODE=xx, \*\*BIND\*\*)
- ◆ The data item names in the list must not include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.
- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.
- ◆ The following is an example of a data item list:

```
*CODE=AB*ORDRCODEORDRCUSTORDRPRODEND.
```



---

Data items omitted from the list and data area are filled with either blanks or nulls, depending on the value of the binary-zero-key field in the database definition. If binary zero keys are allowed, the omitted fields are filled with spaces. If binary zero keys are not allowed, omitted fields are filled with binary zeros.

---

---

**data-area**

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

**Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

---

**endp**

**Description**     *Required.* Delimits the parameter list and indicates the record holding function.

**Format**            4-character field

**Options**            END.                    Holds the record after it is read  
                          RLSE                   Delimits the parameter list (same effect as END.)

## General considerations

- ◆ Data items you do *not* code in the data item list are set to binary zeros.
- ◆ To add a record involving secondary linkpaths, the PDM updates a primary record for each physical key defined for this record. Therefore, these physical keys' identifiers must all be present in the data item list. The data area must contain valid physical key values—they must represent records which exist in the respective primary data sets.
- ◆ If the reference parameter contains LKxx, the record is added at the end of the list.



---

Data items omitted from the list and data area are filled with either blanks or nulls, depending on the value of the binary-zero-key field in the database definition. If binary zero keys are allowed, the omitted fields are filled with spaces. If binary zero keys are not allowed, omitted fields are filled with binary zeros.

---



## ADDVB

Use ADDVB (Add Related Before) to logically add the record in the data area before the record whose RRN is in the reference parameter. This logical addition is made only for the linkpath specified by the linkpath parameter—primary linkpath. For all other linkpaths defined for this record (secondary linkpaths), the addition is made to the end of each respective list.

---

**ADDVB, *status*, *data-set*, *reference*, *linkpath*, *physical-key*,  
*data-item-list*, *data-area*, endp**

---

### ***status***

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### **Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

---

### ***data-set***

**Restriction**     *Required.* Specifies the name of the required data set.

**Format**            4-byte field

**Consideration** The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter

**reference**

---

|                      |                                                                                                                               |                                                                                                                                                                                                           |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>   | <i>Required.</i> Points to a field identifying the RRN of the record after which the new record in this chain is to be added. |                                                                                                                                                                                                           |
| <b>Format</b>        | 4 alphanumeric characters or a 4-byte binary integer                                                                          |                                                                                                                                                                                                           |
| <b>Options</b>       | LKxx                                                                                                                          | The last 4 characters of the linkpath named by the linkpath parameter (where xx = the last 2 characters). The PDM adds the record to the beginning of the linkpath rather than after a particular record. |
|                      | rrrr                                                                                                                          | The RRN of the record after which the new record is to be added to the linkpath.                                                                                                                          |
| <b>Consideration</b> | After successful execution, this parameter contains the RRN of the record just added.                                         |                                                                                                                                                                                                           |

---

## ***linkpath***

**Description**     *Required.* Specifies the name of the linkpath as defined in the compiled database description. All related record functions use this parameter to indicate which related record list is being processed.

**Format**            8-byte field with the following format:

*ppppLKxx*

where:

|             |                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------|
| <i>pppp</i> | represents the name of an associated primary data set.                                        |
| LK          | is a literal (type in as shown).                                                              |
| <i>xx</i>   | represents the last 2 characters of the linkpath name as defined in the database description. |

### **Considerations**

- ◆ The primary data set contains the controlling primary record whose key is given in the primary key parameter. LKxx are the same characters used in the reference parameter to identify the beginning of the list. If you specify an invalid linkpath, the PDM returns MLNF, indicating that the linkpath is not defined in the compiled database description.
- ◆ There are two types of linkpaths: primary and secondary. A primary linkpath is the linkpath specified in add commands—ADDVC, ADDVA, ADDVB and ADDVR. Secondary linkpath refers to all other linkpaths defined for a particular record in the compiled database description.
- ◆ To choose the primary linkpath for a record, you need to know the average length of each record list and the frequency of access along each list. Generally, the primary linkpath should be either the longest record list or the most frequently accessed. When using PDML, *always* use the primary linkpath for ordering records.

---

### ***physical-key***

**Description**     *Required.* Points to a field containing the key of the record in the primary file named in the linkpath parameter.

**Format**            Variable-length as defined on the Directory

**Consideration** During the command processing, if the *physical-key* parameter does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should use the *physical-key* field name in the data area for this parameter, rather than define a separate field.

---

**data-item-list**

**Description**     *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “[Data list parameter keywords](#)” on page 233.)

**Format**            Variable-length field in the following format:

*dataitem1dataitem2...dataitemnEND.*

**Considerations**

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Record codes (ORDRCODE)
  - Special data item processing keywords (\*CODE=xx, \*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.

- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.

**UNIX**

---

Data items omitted from the list and data area are filled with either blanks or nulls, depending on the value of the binary-zero-key field in the database definition. If binary zero keys are allowed, the omitted fields are filled with spaces. If binary zero keys are not allowed, omitted fields are filled with binary zeros.

---

- ◆ The following is an example of a data item list:

```
*CODE=AB*ORDRCODEORDRCUSTORDRPRODEND.
```

---

**data-area**

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

**Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

---

**endp**

|                    |                                                                                                                                                              |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Delimits the parameter list and indicates the record holding function.                                                                      |
| <b>Format</b>      | 4-character field                                                                                                                                            |
| <b>Options</b>     | <div>END.                      Holds the record after it is read</div> <div>RLSE                     Delimits the parameter list (same effect as END.)</div> |

**General considerations**

- ◆ Data items you do *not* code in the data item list are set to binary zeros.

**UNIX**


---

Data items omitted from the list and data area are filled with either blanks or nulls, depending on the value of the binary-zero-key field in the database definition. If binary zero keys are allowed, the omitted fields are filled with spaces. If binary zero keys are not allowed, omitted fields are filled with binary zeros.

---

- ◆ To add a record with secondary linkpaths, the PDM updates a primary record for each physical key defined for this record. Therefore, these physical keys' identifiers must all be present in the data item list. The data area must contain valid physical key values—they must represent records which exist in the respective primary data sets.
- ◆ If the reference parameter indicates the start of the list (LKxx), the record is added at the beginning of the list.

# ADDVC

Use ADDVC (Add Related Continue) to logically add the record in the data area to the end of all linkpaths defined for the record in the compiled database description.

**ADDVC, status, data-set, reference, linkpath, physical-key, data-item-list, data-area, endp**

## status

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### Considerations

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

## data-set

**Restriction**     *Required.* Specifies the name of the required data set.

**Format**            4-byte field

**Consideration** The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter



---

**reference**

|                      |                                                                                                                               |                                                                                                                                                                                                     |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>   | <i>Required.</i> Points to a field identifying the RRN of the record after which the new record in this chain is to be added. |                                                                                                                                                                                                     |
| <b>Format</b>        | 4 alphanumeric characters or a 4-byte binary integer                                                                          |                                                                                                                                                                                                     |
| <b>Options</b>       | LKxx                                                                                                                          | The last 4 characters of the linkpath named by the linkpath parameter (where xx = the last 2 characters). The PDM adds the record to the end of the linkpath rather than after a particular record. |
|                      | rrrr                                                                                                                          | The RRN of the record after which the new record is to be added to the linkpath.                                                                                                                    |
| <b>Consideration</b> | After successful execution, this parameter contains the RRN of the record just added.                                         |                                                                                                                                                                                                     |

**linkpath**

**Description**     *Required.* Specifies the name of the linkpath as defined in the compiled database description. All related record functions use this parameter to indicate which related record list is being processed.

**Format**            8-byte field with the following format:

*ppppLKxx*

where:

|             |                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------|
| <i>pppp</i> | represents the name of an associated primary data set.                                        |
| LK          | is a literal (type in as shown).                                                              |
| <i>xx</i>   | represents the last 2 characters of the linkpath name as defined in the database description. |

**Considerations**

- ◆ The primary data set contains the controlling primary record whose key is given in the primary key parameter. LKxx are the same characters used in the reference parameter to identify the beginning of the list. If you specify an invalid linkpath, the PDM returns MLNF, indicating that the linkpath is not defined in the compiled database description.
- ◆ There are two types of linkpaths: primary and secondary. A primary linkpath is the linkpath specified in add commands—ADDVC, ADDVA, ADDVB and ADDVR. Secondary linkpath refers to all other linkpaths defined for a particular record in the compiled database description.
- ◆ To choose the primary linkpath for a record, you need to know the average length of each record list and the frequency of access along each list. Generally, the primary linkpath should be either the longest record list or the most frequently accessed. When using PDML, *always* use the primary linkpath for ordering records.

---

***physical-key***

- Description**     *Required.* Points to a field containing the key of the record in the primary file named in the linkpath parameter.
- Format**           Variable-length as defined on the Directory
- Consideration**   During the command processing, if the *physical-key* parameter does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should use the *physical-key* field name in the data area for this parameter, rather than define a separate field.

---

***data-item-list***

- Description**     *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “[Data list parameter keywords](#)” on page 233.)
- Format**           Variable-length field in the following format:
- dataitem1dataitem2...dataitemnEND.*

## Considerations

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Record codes (ORDRCODE)
  - Special data item processing keywords (\*CODE=xx, \*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.
- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.

---

**UNIX**

Data items omitted from the list and data area are filled with either blanks or nulls, depending on the value of the binary-zero-key field in the database definition. If binary zero keys are allowed, the omitted fields are filled with spaces. If binary zero keys are not allowed, omitted fields are filled with binary zeros.

---

- ◆ The following is an example of a data item list:

```
*CODE=AB*ORDRCODEORDRCUSTORDRPRODEND.
```

---

***data-area***

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

**Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

**endp**

|                    |                                                                                         |                                                   |
|--------------------|-----------------------------------------------------------------------------------------|---------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Delimits the parameter list and indicates the record holding function. |                                                   |
| <b>Format</b>      | 4-character field                                                                       |                                                   |
| <b>Options</b>     | END.                                                                                    | Holds the record after it is read                 |
|                    | RLSE                                                                                    | Delimits the parameter list (same effect as END.) |

**General considerations**

- ◆ Data items you do *not* code in the data item list are set to binary zeros.



---

Data items omitted from the list and data area are filled with either blanks or nulls, depending on the value of the binary-zero-key field in the database definition. If binary zero keys are allowed, the omitted fields are filled with spaces. If binary zero keys are not allowed, omitted fields are filled with binary zeros.

---

- ◆ To add a record with secondary linkpaths, the PDM updates a primary record for each physical key defined for this record. Therefore, these physical keys' identifiers must all be present in the data item list. The data area must contain valid physical key values—they must represent records which exist in the respective primary data sets.

## ADDVR

Use ADDVR (Add Related Replace) to logically connect an existing related record with different lists without physically moving the record in the data set. The PDM examines every physical key parameter in the data item list and compares its value in the data area with the previous value in the data record.

- ◆ If a physical key parameter has changed, the PDM updates it in the data record and disconnects the record from the old list. SUPRA Server then logically adds the record to the end of the new list controlled by the physical key value in the data area.
- ◆ If a physical key field has not changed, the PDM does not disturb that linkage.

---

**ADDVR, status, data-set, reference, linkpath, physical-key, data-item-list, data-area, endp**

---

### status

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### Considerations

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

**data-set**

---

- Restriction**     *Required.* Specifies the name of the required data set.
- Format**            4-byte field
- Consideration** The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter
- 

**reference**

---

- Description**     *Required.* Points to a field identifying the RRN of the existing record to be compared to your data area.
- Format**            4 alphanumeric characters or a 4-byte binary integer
- Consideration** After successful completion, the RRN remains the same.
- 

**linkpath**

---

- Description**     *Required.* Specifies the name of the linkpath as defined in the compiled database description. All related record functions use this parameter to indicate which related record list is being processed.
- Format**            8-byte field with the following format:
- ppppLKxx*
- where:
- |             |                                                                                               |
|-------------|-----------------------------------------------------------------------------------------------|
| <i>pppp</i> | represents the name of an associated primary data set.                                        |
| LK          | is a literal (type in as shown).                                                              |
| xx          | represents the last 2 characters of the linkpath name as defined in the database description. |



## Considerations

- ◆ The primary data set contains the controlling primary record whose key is given in the primary key parameter. LKxx are the same characters used in the reference parameter to identify the beginning of the list. If you specify an invalid linkpath, the PDM returns MLNF, indicating that the linkpath is not defined in the compiled database description.
- ◆ There are two types of linkpaths: primary and secondary. A primary linkpath is the linkpath specified in add commands—ADDVC, ADDVA, ADDVB and ADDVR. Secondary linkpath refers to all other linkpaths defined for a particular record in the compiled database description.
- ◆ To choose the primary linkpath for a record, you need to know the average length of each record list and the frequency of access along each list. Generally, the primary linkpath should be either the longest record list or the most frequently accessed. When using PDML, *always* use the primary linkpath for ordering records.

---

## *physical-key*

**Description**     *Required.* Points to a field containing the key of the record in the primary file named in the linkpath parameter.

**Format**           Variable-length as defined on the Directory

## Considerations

- ◆ During the command processing, if the *physical-key* parameter does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should use the *physical-key* field name in the data area for this parameter, rather than define a separate field.
- ◆ If you are changing the key value associated with the linkpath parameter, this field must contain the new key value.

**data-item-list**

**Description**     *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “[Data list parameter keywords](#)” on page 233.)

**Format**            Variable-length field in the following format:

`dataitem1dataitem2...dataitemnEND.`

**Considerations**

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Record codes (ORDRCODE)
  - Special data item processing keywords (\*CODE=xx, \*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.

- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.

**UNIX**


---

Data items omitted from the list and data area are filled with either blanks or nulls, depending on the value of the binary-zero-key field in the database definition. If binary zero keys are allowed, the omitted fields are filled with spaces. If binary zero keys are not allowed, omitted fields are filled with binary zeros.

---

- ◆ If you are changing the record's code, this list must name data items that are valid for the new code.
- ◆ You can update the value of any of the data items in addition to updating control keys or record code.
- ◆ The following is an example of a data item list:

```
*CODE=AB*ORDRCODEORDRCUSTORDRPRODEND.
```

---

**data-area**

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**             The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

**Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

**endp**

**Description**     *Required.* Points to a field that delimits the parameter list.

**Format**            4-character field

**Options**            END.                    Holds the record after it is read

                         RLSE                    Holds the record after it is read

**General considerations**

- ◆ ADDVR is the only command that can change the record code. This change could add or delete a linkpath in the record. The PDM examines the linkpaths defined in the SUPRA Server compiled database description for the old and new record codes.

Any linkpaths not common to both codes are deleted or added as required. Additions are made to the logical end of the list controlled by the physical key value in the data area.

- ◆ If you use the ADDVR command to change the record code, code all data items which occur in the redefined area for the new record code, even though they are not being changed.

## CNTRL (VMS only)

Use CNTRL to hold single or multiple records.

---

**CNTRL *status, options, area-length, data-item-list, data-area, endp***

---

---

### ***status***

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### **Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

**options**

**Description**     *Required.* Specifies a list of parameters concerned with holding single or multiple records.

**Format**           Variable-length field

**Options**           ACTION=SET or FREE  
                      *Required.* Specifies whether to hold or release a record.

TYPE=READ,LOCK or ANYLOCK

*Required.* Identifies the type of hold to be obtained or released. READ is equivalent to shared read hold, LOCK is equivalent to exclusive hold. ANYLOCK can be used only if the ACTION=FREE, releasing both shared read holds and exclusive holds.

LEVEL=FILE or RECORD

*Optional.* Specifies the level (file level or record level) at which the hold applies.

FILE=*data-set-name*

*Required.* Identifies the file containing the record(s) to hold.

TIME=*seconds*

*Optional.* Specifies the maximum time to wait for a record hold in numbers of seconds. The value you specify for TIME takes precedence over the value specified for DEFTIME.

DEFTIME=*seconds*

*Optional.* Specifies the default time to wait for a record hold. This time is used if you omit the TIME option and applies to all DML functions trying to acquire record holds. You can change the default at any time while a task is signed on. The TIME value takes precedence over the DEFTIME value.

**Considerations**

- ◆ You can specify any or all of the option parameters separated by commas and terminated by END.
- ◆ You can specify each option more than once, but the PDM uses only the last occurrence.
- ◆ The following is an example of an option parameter:

ACTION=SET,TYPE=LOCK,LEVEL=RECORD,FILE=PART, TIME=0001,END.

---

**area-length**

|                    |                                                                                                                                                                                                                    |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Specifies the length of the data area containing the list of RRNs or key values. If an error condition such as HELD occurs, the PDM returns the relative position of the key or RRN which failed. |
| <b>Format</b>      | 4-byte binary integer                                                                                                                                                                                              |

---

**data-item-list**

|                    |                                                                                                                                    |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Specifies data items to moved to or from the data area. This list serves as a map of the layout of the data area. |
| <b>Format</b>      | 12-character field containing one of the following keywords plus END.:                                                             |
| *KEYLIST           | Indicates that the data area contains a list of control keys (primary files only). You must use the LEVEL=RECORD option.           |
| *RRNLIST           | Indicates that the data area contains a list of RRNs. You must use the LEVEL=RECORD option.                                        |
| **ALL.**           | All records. You must use the LEVEL=FILE option.                                                                                   |
| **NONE**           | No records. No action is taken.                                                                                                    |

**Consideration** Always terminate this parameter with END.

---

**data-area**

|                    |                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> An area containing a list of RRNs or key values to be held.                                                                                                                                                                                                                                                                                           |
| <b>Format</b>      | Variable-length area of sufficient size to hold one of the following: <ul style="list-style-type: none"> <li>◆ If the data list contains *RRNLIST—an array of 4-byte binary integer RRNs to be held.</li> <li>◆ If the data list contains *KEYLIST—an array of control-key-size character fields containing keys to the primary file of records to be held.</li> </ul> |

**Consideration** This field is not examined if the data list contains \*\*NONE\*\* or \*\*ALL.\*\*\*, but the field must point to a valid parameter area.

endp

|             |                                                                                  |                                      |  |
|-------------|----------------------------------------------------------------------------------|--------------------------------------|--|
| Description | Required. Delimits the parameter list and indicates the record holding function. |                                      |  |
| Format      | 4-character field                                                                |                                      |  |
| Options     | END.                                                                             | Holds the record after it is read    |  |
|             | RLSE                                                                             | Releases the record after it is read |  |

General considerations

- ◆ A task can obtain one of two types of hold on a record or file:
  - LOCK obtains an exclusive hold
  - READ obtains a shared read hold
- ◆ No two tasks may LOCK the same record or file concurrently; the second task receives a HELD status. However, two tasks may obtain a shared hold on a record or file as shown in the following table.

| Hold requested by task 2 | Existing hold obtained by task 1 |             |           |           |
|--------------------------|----------------------------------|-------------|-----------|-----------|
|                          | Lock record                      | Read record | Lock file | Read file |
| Lock file                | HELD                             | HELD        | HELD      | HELD      |
| Read file                | HELD                             | ****        | HELD      | ****      |
| Lock record              | HELD                             | HELD        | HELD      | HELD      |
| Read record              | HELD                             | ****        | HELD      | ****      |

- ◆ For example, assume TASK1 has a shared read on FILE-A (the last column in the table.) If TASK2 attempts to lock FILE-A, it will receive a HELD status. If, however, TASK2 attempts a shared read of FILE-A, it receives a successful status \*\*\*\* and can continue processing.
- ◆ The default hold level is RECORD (see the description of the Option parameter).
- ◆ Any value specified for the TIME option takes precedence over any value specified for the DEFTIME option. Any value specified for DEFTIME takes precedence over the RETRY value specified as a PDM input parameter. (Refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130.)



## COMIT

Use COMIT to inform the PDM that the current task has completed a logical unit of work. It also indicates that updates performed since the last COMIT are permanent (no longer subject to back-out via RESET or because of a system or program failure).

COMIT also allows you to write a comment in the COMIT record. By executing COMIT, you momentarily cause SUPRA Server to:

- ◆ Synchronize all processing
- ◆ Physically write any buffers affected by this task to disk
- ◆ Release all resources held by this task
- ◆ (*If task logging is inactive*) Reset the task log file area for this task

### UNIX

---

All read-ahead buffers are released.

---

If you issue COMIT and task logging is inactive, the buffers are physically written, resources are released and processing continues.

---

**COMIT, *status*, *area-length*, *data-area*, endp**

---

**status**

**Description** *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format** 4-character field

**Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

---

**area-length**

**Description** *Required.* Specifies to the PDM the length of the data stored in the data area that will be written to the log file.

**Format** 4-byte binary integer

**Consideration** If no data is to be stored by COMIT, the area-length may be set to zero.

---

**data-area**

**Description** *Required.* An area containing the data to be written to the log file.

**Format** Variable-length field equal to the value specified in the area-length parameter.

**Consideration** The amount of data stored cannot exceed the size of the log file.

---

**endp**

|                    |                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Points to a field that delimits the parameter list.                                             |
| <b>Format</b>      | 4-character field                                                                                                |
| <b>Options</b>     | END. Delimits the parameter list (same effect as RLSE)<br>RLSE Delimits the parameter list (same effect as END.) |

**General considerations**

- ◆ COMIT immediately releases all records held by the task so other tasks may access those records.
- ◆ Excessive use of COMIT affects performance since additional I/O is required.
- ◆ The area length should not exceed the task log buffer size minus 72 or an LSZE status is returned. If this occurs, the COMIT is not completed.
- ◆ If you issue a RESET, the data written to the task log file by COMIT is returned to the application program. This data identifies the COMIT point to which the program has been reset.

## DEL-M

Use DEL-M to delete a primary record by setting it to binary zeros. The unused location is available for immediate reuse.

---

**DEL-M, status, data-set, physical-key, data-item-list, data-area, endp**

---

---

### status

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

**Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

---

### data-set

**Restriction**     *Required.* Specifies the name of the required data set.

**Format**            4-byte field

**Consideration** The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter

---

**physical-key**

|                    |                                                                                                                                                        |
|--------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Points to a field containing the key of the primary record to be processed. The PDM uses this parameter to locate the primary record. |
| <b>Format</b>      | Variable-length as defined on the Directory                                                                                                            |

---

**data-item-list**

|                    |                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> This field is not examined; however, it must point to a valid parameter area. |
|--------------------|------------------------------------------------------------------------------------------------|

---

**data-area**

|                    |                                                                                                |
|--------------------|------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> This field is not examined; however, it must point to a valid parameter area. |
|--------------------|------------------------------------------------------------------------------------------------|

---

**endp**

|                    |                                                                                                                                     |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Points to a field that delimits the parameter list.                                                                |
| <b>Format</b>      | 4-character field                                                                                                                   |
| <b>Options</b>     | <p>END.     Delimits the parameter list (same effect as RLSE)</p> <p>RLSE     Delimits the parameter list (same effect as END.)</p> |

**General considerations**

- ◆ The PDM does not delete a primary record if any related records remain linked to it. You must first use DELVD to delete each related record.
- ◆ Because of internal space management considerations, you should not use this command with the serial retrieval command RDNXT. If you do, records might be read twice or missed completely during the serial read process.

## DELVD

Use DELVD to delete the related record whose RRN is in the reference parameter. The unused location is available for immediate reuse. The record is disconnected from all associated primary records. Upon completion of DELVD, the reference parameter returned contains the RRN of the record that logically precedes the deleted record, or LKxx if there is no preceding record.

---

**DELVD, status, data-set, reference, linkpath, physical-key, data-item-list, data-area, endp**

---

---

### status

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

#### Considerations

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

---

**data-set**

- Restriction**     *Required.* Specifies the name of the required data set.
- Format**           4-byte field
- Consideration**   The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter.

---

**reference**

- Description**     *Required.* Points to a field identifying the RRN of the related record to be deleted.
- Format**           4 alphanumeric characters or a 4-byte binary integer
- Consideration**   When the DELVD command completes, the PDM updates the reference field to the RRN of the *previous* logical record in the chain. However, if the deleted record was logically the first record on the chain, the reference field is updated to LKxx instead (where xx = the actual characters from the linkpath name).

**linkpath**

**Description**     *Required.* Specifies the name of the linkpath as defined in the compiled database description. All related record functions use this parameter to indicate which related record list is being processed.

**Format**            8-byte field with the following format:

*ppppLKxx*

where:

*pppp*                represents the name of an associated primary data set.

LK                   is a literal (type in as shown).

*xx*                   represents the last 2 characters of the linkpath name as defined in the database description.

**Considerations**

- ◆ The primary data set contains the controlling primary record whose key is given in the primary key parameter. LKxx are the same characters used in the reference parameter to identify the beginning of the list. If you specify an invalid linkpath, the PDM returns MLNF, indicating that the linkpath is not defined in the compiled database description.
- ◆ There are two types of linkpaths: primary and secondary. A primary linkpath is the linkpath specified in add commands—ADDVC, ADDVA, ADDVB and ADDVR. Secondary linkpath refers to all other linkpaths defined for a particular record in the compiled database description.
- ◆ To choose the primary linkpath for a record, you need to know the average length of each record list and the frequency of access along each list. Generally, the primary linkpath should be either the longest record list or the most frequently accessed. When using PDML, *always* use the primary linkpath for ordering records.



---

**physical-key**

**Description**     *Required.* Points to a field containing the key of the record in the primary file named by the linkpath parameter. Many other primary records may need updating as a result of this DELVD command.

**Format**            Variable-length as defined on the Directory

**Considerations**

- ◆ If the physical key value does not match the value in the record to be deleted (RRN), the PDM returns an error status code .
- ◆ This parameter specifies the key of the controlling primary record.
- ◆ The PDM determines a Relative Record Number (RRN) for each record by performing a calculation on the physical key's value. This process is the relative address calculator. The PDM uses the RRN to add or read the primary record. The length of the physical key for the relevant primary data set is defined in the compiled database description. When you are performing a related record function, this parameter specifies the key of the controlling primary record.

---

**data-item-list**

**Description**     *Required.* This field is not examined; however, it must point to a valid parameter area.

---

**data-area**

**Description**     *Required.* This field is not examined; however, it must point to a valid parameter area.

**endp**

|                    |                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Points to a field that delimits the parameter list.                                             |
| <b>Format</b>      | 4-character field                                                                                                |
| <b>Options</b>     | END. Delimits the parameter list (same effect as RLSE)<br>RLSE Delimits the parameter list (same effect as END.) |

**General considerations**

- ◆ If you perform a READV immediately after a DELVD without changing the reference parameter, SUPRA Server retrieves the record logically following the deleted record.
- ◆ If you issue a READD immediately after a DELVD without changing the reference parameter, SUPRA Server retrieves the record logically preceding the deleted record.
- ◆ If you issue a READR immediately after a DELVD without changing the reference parameter, the *record logically preceding the deleted record is skipped*, and the next preceding record is retrieved.
- ◆ To delete an entire list of records, read the last one (READR) with record holding. Then perform successive DELVD commands until the reference parameter is set to LKxx.

## MARKL

Use MARKL (Mark the System Log File with User Record) to enable the application program to write records containing your own information onto the System Log File. These MARKL records appear in the System Log File together with the function records the PDM creates. Your records are ignored when the system log is processed by the SUPRA Server recovery program.




---

All read-ahead buffers are released.

---



---

**MARKL, *status*, *area-length*, *data-area*, *endp***

---

### ***status***

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### **Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

---

**area-length**

|                    |                                                                                                                            |
|--------------------|----------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Specifies to the PDM the length of the data stored in the data area that will be written to the log file. |
| <b>Format</b>      | 4-byte binary integer                                                                                                      |

---

**data-area**

|                      |                                                                                  |
|----------------------|----------------------------------------------------------------------------------|
| <b>Description</b>   | <i>Required.</i> An area containing the data to be written to the log file.      |
| <b>Format</b>        | Variable-length field equal to the value specified in the area-length parameter. |
| <b>Consideration</b> | The amount of data stored cannot exceed the size of the log file.                |

---

**endp**

|                    |                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Points to a field that delimits the parameter list.                                                     |
| <b>Format</b>      | 4-character field                                                                                                        |
| <b>Options</b>     | END.     Delimits the parameter list (same effect as RLSE)<br>RLSE     Delimits the parameter list (same effect as END.) |

**General consideration**

Excessive use of MARKL affects performance since MARKL writes a user record to the system log, thus requiring additional I/O. Programs will run more efficiently if you use MARKL only when necessary.

## OPCOM

Use OPCOM to pass requests to and receive output from the PDM, in effect programming your own interface to the PDM instead of using CSIOPCOM, VMS REPLY, or UNIX csireply.

You can use this function either on its own or in parallel with the existing VMS REPLY or UNIX csireply facility. Refer to the *SUPRA Server PDM System Administration Guide (VMS)*, P25-0130, for a description of how to suppress the VMS REPLY or UNIX csireply facility using the PDM input parameter SYSOPCOM=N.

---

**OPCOM *status, qualifier, node-name, command, area-length, data-area, endp***

---

### *status*

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### **Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

---

**qualifier**

|                    |                                                                                                                                                                                                                |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> A field which receives a return value of NEXT if the specified PDM operator command returns more than one line. A value of END. is returned when no additional lines of output are available. |
| <b>Format</b>      | 4-byte field                                                                                                                                                                                                   |

---

**node-name**

|                    |                                                                                                                                                                                                                                  |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Specifies where the PDM is running.                                                                                                                                                                             |
| <b>Format</b>      |                                                                                                                                                                                                                                  |
| <b>VMS</b>         | A 6-character field containing the logical name of the VMS node on which the command should be executed. Pad the field with spaces if the node name is less than six characters. Fill the field with spaces if the PDM is local. |
| <b>UNIX</b>        | A character field containing the node name of the machine on which the PDM is running. This field must be a null terminated string.                                                                                              |

---

**command**

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>    | <i>Required.</i> Contains the PDM operator command to be executed by the OPCOM function.                                                                                                                                                                                                                                                                                                                                          |
| <b>Format</b>         | Variable-length field                                                                                                                                                                                                                                                                                                                                                                                                             |
| <b>Considerations</b> | <ul style="list-style-type: none"><li>◆ Always terminate this parameter with END.</li><li>◆ The following is an example:<br/><pre>DISPLAY/DATABASES END.<br/>UNLOAD/FORCE BURRYS[000114]END.</pre></li><li>◆ For details on valid PDM operator commands, refer to the <i>SUPRA Server PDM System Administration Guide (UNIX)</i>, P25-0132; or the <i>SUPRA Server PDM System Administration Guide (VMS)</i>, P25-0130.</li></ul> |

---

**area-length**

|                    |                                                                                        |
|--------------------|----------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> The PDM sets this parameter to the length of the PDM output returned. |
| <b>Format</b>      | 4-byte binary integer                                                                  |

---

**data-area**

|                      |                                                                                                                                                                                   |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>   | <i>Required.</i> An area to contain any output data returned to your program by the OPCOM command.                                                                                |
| <b>Format</b>        | A variable-length data area of sufficient length to hold the returned data (maximum = 132 characters)                                                                             |
| <b>Consideration</b> | SUPRA PDM returns data to your program as a formatted report with line lengths of up to 132 characters. Each call returns one line; therefore, the maximum data area size is 132. |

**endp**

|                    |                                                                                                                          |
|--------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Points to a field that delimits the parameter list.                                                     |
| <b>Format</b>      | 4-character field                                                                                                        |
| <b>Options</b>     | END.     Delimits the parameter list (same effect as RLSE)<br>RLSE     Delimits the parameter list (same effect as END.) |

**General considerations**

- ◆ OPCOM does not have any automatic PDM startup or restart capability.
- ◆ If the PDM input file parameter **CONSOLE** is set to **YES**, the Physical DML OPCOM command sends the result to the specified OPERATOR terminal as well as to the DML function data area. If the PDM input file parameter **CONSOLE** is set to **NO**, OPCOM sends the result to the DML function data area only.

**VMS**

---

You can suppress output to the OPERATOR terminal without setting SYSOPCOM to NO by disabling the specified OPERATOR terminal.

---

- ◆ When multiple lines of output are generated by the OPCOM command (such as with the **DISPLAY/DATABASES** command), the qualifier parameter will contain **NEXT**. This indicates that additional OPCOM commands are necessary to retrieve additional lines of output. When the qualifier contains **END.**, no additional lines of output are available for the OPCOM command executed. The qualifier must not contain **END.** before the call to **datbas** or no data will be returned by the PDM. Be sure the value **END.** is not retained in the qualifier parameter from a previous call.
- ◆ An application should not be signed on to a database when using the OPCOM function call. OPCOM is a special DML function and does not operate properly when an application is signed on, that is, when an application has successfully performed a **SINON** function without a **SINOF** function. If an application is signed on to a database when the OPCOM function is issued, then it is possible for the application to receive a success status of "\*\*\*\*\*" while receiving no data.



## RDNXT

Use RDNXT (Read Next) to provide a generalized serial retrieval method. You can direct the retrieval to a specific point in the data set (such as the beginning of the data set) or a specific record location (RRN) within the data set. Each retrieved record is placed in the data area. If a coded data list is used, record codes omitted from the list are not retrieved.

You can continue retrieval by simply reexecuting RDNXT until you reach the end of the data set. Only data records are returned to the application program; unused records and control records are bypassed. The RRN of the record read is returned in the qualifier. Subsequent executions of RDNXT can continue processing serially from that point. When the end of the data set is reached, the characters END. are returned in the qualifier.

See “[Structural maintenance during serial processing](#)” on page 362 and “[Understanding RDNXT serial processing](#)” on page 367 for more information on RDNXT.

---

**RDNXT, status, data-set, qualifier, data-item-list, data-area, endp**

---

### status

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### Considerations

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the [SUPRA Server PDM Messages and Codes Reference Manual \(PDM/RDM Support for UNIX & VMS\)](#), P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

**data-set**

- Restriction**     *Required.* Specifies the name of the required data set.
- Format**         4-byte field
- Consideration** The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter.

**qualifier**

- Description**     *Required.* Specifies parameters concerned with serial processing of records.
- Format**           4-byte field
- Options**          The following table lists the qualifier values and their meanings when used with RDNXT.

| Value                                                                                      | Meaning when passed to the PDM                                                                                      | Meaning when returned by the PDM                                                                                    |
|--------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------|
| BEGN (four ASCII characters)                                                               | Read the existing record with the lowest RRN (Relative Record Number) in the indicated primary or related data set. | n/a                                                                                                                 |
| rrrr (The RRN of an existing record in the current data set expressed as a 4-byte integer) | Read the record with the next higher RRN than the one supplied.                                                     | Identifies the RRN of the record read.                                                                              |
| END. (four ASCII characters)                                                               |                                                                                                                     | No record was read because no record exists in the current data set with a higher RRN value than the RRN specified. |

- Consideration** To read the first record in the data set, set the field to BEGN. For each RDNXT, the PDM returns a value in this field identifying the record just read by SUPRA Server. Do not change this value; use it only for the following purposes:
- ◆ To test whether all the records in the data set have been read
  - ◆ To read the next record
- When all the records have been read, SUPRA Server sets the qualifier to END. and the returned data area is no longer valid. You can save the qualifier value to continue reading from the current position later.

---

**data-item-list**

**Description**     *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “[Data list parameter keywords](#)” on page 233.)

**Format**            Variable-length field in the following format:  
                          `dataitem1dataitem2...dataitemnEND.`

**Considerations**

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Physical keys (CUSTCTRL)
  - Record codes (ORDRCODE)
  - Special data item processing keywords (\*CODE=xx, \*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.
- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.
- ◆ The following are 2 examples of data item lists:

```
CUSTCTRLCUSTADDRCUSTCREND.
```

```
BINDCODE=AB*ORDRCODEORDRCUSTORDRPRODEND.
```

## ***data-area***

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

### **Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

---

**endp**

|                    |                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Delimits the parameter list and indicates the record holding function.     |
| <b>Format</b>      | 4-character field                                                                           |
| <b>Options</b>     | END.     Holds the record after it is read<br>RLSE     Releases the record after it is read |

**General considerations**

- ◆ Because of internal space management considerations, you should not use DEL-M while you are processing primary data sets with RDNXT. DEL-M performs internal space management operations which can physically move other records in the primary data set. Therefore, if DEL-M is performed in conjunction with RDNXT processing, records might be read twice or missed completely during the serial read process.
- ◆ You can perform a WRITM while processing primary data sets with the RDNXT.
- ◆ You can perform WRITV and DELVD while processing related data sets with the RDNXT command. The reference parameters for these commands may be taken from the qualifier parameter.
- ◆ RDNXT performs a physical serial read and does not process according to physical key sequence.

## READD

Use READD (Read Direct) to directly retrieve the related record specified by the RRN in the reference parameter. You can continue any related data set processing, using any valid linkpath for that record.

---

**READD, *status*, *data-set*, *reference*, *linkpath*, *physical-key*,  
*data-item-list*, *data-area*, endp**

---

---

### ***status***

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### **Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

**data-set**

---

- Restriction**     *Required.* Specifies the name of the required data set.
- Format**           4-byte field
- Consideration** The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter.

---

**reference**

- Description**     *Required.* Points to a field containing the RRN of the related record to be read.
- Format**           4 alphanumeric characters or a 4-byte binary integer
- Consideration** When the READD command completes successfully, the reference parameter still contains the RRN of the retrieved record—the PDM does not change reference.

**linkpath**

**Description**     *Required.* Specifies the name of the linkpath as defined in the compiled database description. All related record functions use this parameter to indicate which related record list is being processed.

**Format**            8-byte field with the following format:

*ppppLKxx*

where:

*pppp* represents the name of an associated primary data set.

LK is a literal (type in as shown).

*xx* represents the last 2 characters of the linkpath name as defined in the database description.

**Considerations**

- ◆ The primary data set contains the controlling primary record whose key is given in the primary key parameter. LKxx are the same characters used in the reference parameter to identify the beginning of the list. If you specify an invalid linkpath, the PDM returns MLNF, indicating that the linkpath is not defined in the compiled database description.
- ◆ There are two types of linkpaths: primary and secondary. A primary linkpath is the linkpath specified in add commands—ADDVC, ADDVA, ADDVB and ADDVR. Secondary linkpath refers to all other linkpaths defined for a particular record in the compiled database description.
- ◆ To choose the primary linkpath for a record, you need to know the average length of each record list and the frequency of access along each list. Generally, the primary linkpath should be either the longest record list or the most frequently accessed. When using PDML, always use the primary linkpath for ordering records.

---

**physical-key**

**Description**     *Required.* Points to a field containing the key of the primary record controlling the chain. The PDM uses this key to link from a primary record to a related record.

**Format**            Variable-length as defined on the Directory



---

## ***data-item-list***

**Description**     *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “[Data list parameter keywords](#)” on page 233.)

**Format**            Variable-length field in the following format:

```
dataitem1dataitem2...dataitemnEND.
```

### **Considerations**

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Physical keys (CUSTCTRL)
  - Record codes (ORDRCODE)
  - Special data item processing keywords (\*CODE=xx, \*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.
- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.
- ◆ The following are two examples of data item lists:

```
CUSTCTRLCUSTADDRCUSTCREND.
```

```
*CODE=AB*CODE=XXORDRCUSTORDRPRODEND.
```

## ***data-area***

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

### **Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

---

**endp**

|                    |                                                                                             |
|--------------------|---------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Delimits the parameter list and indicates the record holding function.     |
| <b>Format</b>      | 4-character field                                                                           |
| <b>Options</b>     | END.     Holds the record after it is read<br>RLSE     Releases the record after it is read |

**General considerations**

- ◆ An RRN supplied to READD can come from one of these sources:
  - Your program saved the RRN when processing a related data set, and you are now preparing to resume processing by retrieving a previously read record.
  - A DELVD has just been performed and you wish to retrieve the record preceding the deleted record. In this case, your program must have held the record from the time you obtained the RRN until the time you reused it. Otherwise, another task might have changed, moved or deleted the record.
- ◆ If a coded data list is used and the record code of the record read does not match the code specified in the data list, the error IVRC will be returned in the status parameter.

## READM

Use READM (Read Primary) to read a primary record and write it to the data area according to the data item list. The PDM locates the specified record by using the relative address calculator on the unique physical key.

---

**READM, *status*, *data-set*, *physical-key*, *data-item-list*, *data-area*,  
endp**

---

---

### *status*

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

#### **Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

---

### *data-set*

**Restriction**     *Required.* Specifies the name of the required data set.

**Format**            4-byte field

**Consideration** The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter

---

**physical-key**

|                    |                                                                                                                                                               |
|--------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Points to a field containing the key of the primary record to be processed. The PDM uses this key to identify the primary record being read. |
| <b>Format</b>      | Variable-length as defined on the Directory                                                                                                                   |

---

**data-item-list**

|                    |                                                                                                                                                                                                                                                                                                                                                                                        |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “ <a href="#">Data list parameter keywords</a> ” on page 233.) |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|               |                                                                                                   |
|---------------|---------------------------------------------------------------------------------------------------|
| <b>Format</b> | Variable-length field in the following format:<br><code>dataitem1dataitem2...dataitemnEND.</code> |
|---------------|---------------------------------------------------------------------------------------------------|

**Considerations**

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Physical keys (CUSTCTRL)
  - Special data item processing keywords (\*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.
- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.
- ◆ The following is an example of a data item list:

```
CUSTCTRLCUSTADDRCUSTCREND.
```

**data-area**

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

**Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

---

**endp**

**Description**     *Required.* Delimits the parameter list and indicates the record holding function.

**Format**            4-character field

**Options**            END.     Holds the record after it is read  
                         RLSE     Releases the record after it is read

**General consideration**

To determine whether a primary record exists, read the record using READM. A status of \*\*\*\* indicates that it does exist. A status of MRNF indicates that it does not exist.

## READR

Use READR (Read Reverse) to read the record preceding the specified related record within the defined linkpath. To read an entire list of records in the reverse direction, start processing by placing LKxx into the reference field, and then issue READR. The last record in the list is then returned.

Continue processing by reissuing READR without changing the reference parameter. READR retrieves records in reverse order until the first record of the list has been processed. When you issue a READR with the reference parameter containing the RRN of the first record, END. is returned in the reference parameter to indicate that the list has been completely processed.

---

**READR, *status*, *data-set*, *reference*, *linkpath*, *physical-key*, *data-item-list*, *data-area*, endp**

---



---

### ***status***

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### **Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

---

### ***data-set***

**Restriction**     *Required.* Specifies the name of the required data set.

**Format**         4-byte field

**Consideration** The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter.



---

**reference**

**Description**     *Required.* Points to a field identifying the position in a related record chain. You use the reference parameter by placing a certain value in this field to tell the PDM which record to begin with. The PDM returns a certain value to inform you which related record was processed.

**Format**            4-byte field

**Options**            LK<sub>xx</sub> The last 4 characters of the linkpath named by the linkpath parameter (where xx = the last 2 characters). This value directs the PDM, for READR, to read the last logical record in the chain for this control key.

rrrr This is set by the PDM to identify the RRN of the record just read. When you issue a READR with reference still set to rrrr from the previous READR, the PDM uses this RRN to locate the preceding logical record in the chain.

**Considerations**

- ◆ If the reference parameter contains an RRN, the READR uses the back pointer in that record to find the RRN of the preceding record. The preceding record is retrieved and its RRN is placed in the reference field. Therefore, reference always contains the RRN of the record just read.
- ◆ The PDM places the keyword END. in the reference field if you attempt to go beyond the end of the chain. This signifies that the previous read returned the first logical record in the chain (head-of-chain). You can reinitialize the parameters to begin another chain process.
- ◆ If the PDM sets END. for a first READR execution with LK<sub>xx</sub>, the chain is empty.
- ◆ Because END. is not valid as input reference for READR, your program must change this value before it issues another READR. You must, therefore, include logic in your program to test the reference field for END. after each command to detect the logical end of the search (beginning of the chain).

**linkpath**

**Description**     *Required.* Specifies the name of the linkpath as defined in the compiled database description. All related record functions use this parameter to indicate which related record list is being processed.

**Format**            8-byte field with the following format:

*ppppLKxx*

where:

*pppp*                represents the name of an associated primary data set.

LK                   is a literal (type in as shown).

*xx*                   represents the last 2 characters of the linkpath name as defined in the database description.

**Considerations**

- ◆ The primary data set contains the controlling primary record whose key is given in the primary key parameter. LKxx are the same characters used in the reference parameter to identify the beginning of the list. If you specify an invalid linkpath, the PDM returns MLNF, indicating that the linkpath is not defined in the compiled database description.
- ◆ There are two types of linkpaths: primary and secondary. A primary linkpath is the linkpath specified in add commands—ADDVC, ADDVA, ADDVB and ADDVR. Secondary linkpath refers to all other linkpaths defined for a particular record in the compiled database description.
- ◆ To choose the primary linkpath for a record, you need to know the average length of each record list and the frequency of access along each list. Generally, the primary linkpath should be either the longest record list or the most frequently accessed. When using PDML, *always* use the primary linkpath for ordering records.

---

**physical-key**

**Description**     *Required.* Points to a field containing the key of the primary record controlling the chain. The PDM uses this key to link from a primary record to a related record.

**Format**            Variable-length as defined on the Directory

## ***data-item-list***

**Description**     *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “[Data list parameter keywords](#)” on page 233.)

**Format**            Variable-length field in the following format:

*dataitem1dataitem2...dataitemnEND.*

### **Considerations**

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Record codes (ORDRCODE)
  - Special data item processing keywords (\*CODE=xx, \*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.
- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.
- ◆ The following is an example of a data item list:

**\*\*BIND\*\*\*CODE=AB\*ORDRCODEORDRCUSTORDRPRODEND.**

**data-area**

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

**Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

---

**endp**

**Description**     *Required.* Delimits the parameter list and indicates the record holding function.

**Format**            4-character field

**Options**            END.     Holds the record after it is read  
                         RLSE     Releases the record after it is read

**General consideration**

When the reference parameter contains END. and the list of records has been completely processed, any further related data set commands executed return an IVRP (Invalid Reference Parameter) status code.

## READV

Use READV (Read Related) to read the record following the specified related record within the defined linkpath. To read an entire list of records, initiate processing by placing LKxx into the reference parameter and issue READV. The first record of the list is then returned.

Continue processing by reissuing READV without changing the reference parameter. READV retrieves records in forward order until the last record in the list has been processed. When you issue a READV with the reference parameter containing the RRN of the last record, END. is returned in the reference parameter to indicate that the list has been completely processed.

---

**READV, status, data-set, reference, linkpath, physical-key, data-item-list, data-area, endp**

---



---

### status

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### Considerations

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

**data-set**

|                      |                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Restriction</b>   | <i>Required.</i> Specifies the name of the required data set.                                                                                         |
| <b>Format</b>        | 4-byte field                                                                                                                                          |
| <b>Consideration</b> | The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter. |

**reference**

|                    |                                                                                                                                                                                                                                                                                               |                                                                                                                                                                                                                            |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Points to a field identifying the position in a related record chain. You use the reference parameter by placing a certain value in this field to tell the PDM which record to begin with. The PDM returns a certain value to inform you which related record was processed. |                                                                                                                                                                                                                            |
| <b>Format</b>      | 4 -byte field                                                                                                                                                                                                                                                                                 |                                                                                                                                                                                                                            |
| <b>Options</b>     | LKxx                                                                                                                                                                                                                                                                                          | The last 4 characters of the linkpath named by the linkpath parameter (where xx = the last 2 characters). This value directs the PDM, for READV, to read the last logical record in the chain for this control key.        |
|                    | rrrr                                                                                                                                                                                                                                                                                          | This is set by the PDM to identify the RRN of the record just read. When you issue a READV with reference still set to rrrr from the previous READV, the PDM uses this RRN to locate the next logical record in the chain. |

**Considerations**

- ◆ If the reference parameter contains an RRN, the READV uses the forward pointer in that record to find the RRN of the next record. The next record is retrieved and its RRN is placed in the reference field. Therefore, reference always contains the RRN of the record just read.
- ◆ The PDM places the keyword END. in the reference field if you attempt to go beyond the end of the chain. This signifies that the previous read returned the last logical record in the chain (end-of-chain). You can reinitialize the parameters to begin another chain process.
- ◆ If the PDM sets END. for a first READV execution with LKxx, the chain is empty.
- ◆ Since END. is not valid as input reference for READV your program must change this value before it issues another READV. You must, therefore, include logic in your program to test the reference field for END. after each command to detect the logical end of the search (end of the chain).

---

**linkpath**

**Description**     *Required.* Specifies the name of the linkpath as defined in the compiled database description. All related record functions use this parameter to indicate which related record list is being processed.

**Format**            8-byte field with the following format:

*ppppLKxx*

where:

*pppp*                represents the name of an associated primary data set.

LK                    is a literal (type in as shown).

xx                    represents the last 2 characters of the linkpath name as defined in the database description.

**Considerations**

- ◆ The primary data set contains the controlling primary record whose key is given in the primary key parameter. LKxx are the same characters used in the reference parameter to identify the beginning of the list. If you specify an invalid linkpath, the PDM returns MLNF, indicating that the linkpath is not defined in the compiled database description.
- ◆ There are two types of linkpaths: primary and secondary. A primary linkpath is the linkpath specified in add commands—ADDVC, ADDVA, ADDVB and ADDVR. Secondary linkpath refers to all other linkpaths defined for a particular record in the compiled database description.
- ◆ To choose the primary linkpath for a record, you need to know the average length of each record list and the frequency of access along each list. Generally, the primary linkpath should be either the longest record list or the most frequently accessed. When using PDML, *always* use the primary linkpath for ordering records.

---

**physical-key**

**Description**     *Required.* Points to a field containing the key of the primary record controlling the chain. The PDM uses this key to link from a primary record to a related record.

**Format**            Variable-length as defined on the Directory

**data-item-list**

**Description**     *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “[Data list parameter keywords](#)” on page 233.)

**Format**            Variable-length field in the following format:

*dataitem1dataitem2...dataitemnEND.*

**Considerations**

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Record codes (ORDRCODE)
  - Special data item processing keywords(\*CODE=xx, \*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.
- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.
- ◆ The following is an example of a data item list:

*\*\*BIND\*\*CODE=AB\*ORDRCODEORDRCUSTORDRPRODEND.*



---

**data-area**

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

**Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

---

**endp**

**Description**     *Required.* Delimits the parameter list and indicates the record holding function.

**Format**            4-character field

**Options**            END.     Holds the record after it is read

                         RLSE     Releases the record after it is read

**General consideration**

When the reference parameter contains END. and the list of records has been completely processed, any further related data set commands return an IVRP (Invalid Reference Parameter) status code.

# READX

Use READX (Read Index) to read either a primary or related record and write it to the data area according to the data item list. The PDM locates the record using the index specified in the OPTIONS parameter.

Use BEGN in the QUALIFIER parameter to provide direct access to a record using the full or partial key supplied in the QUALIFIER parameter. The PDM automatically replaces BEGN with NEXT.

Continue processing by reissuing READX without changing the qualifier parameter. READX retrieves records in ascending or descending order depending on the direction specified in the OPTIONS parameter. When the QUALIFIER parameter contains END., the last record in the data set has been processed.

**READX *status, data-set, options, qualifier, area-length, data-item-list, data-area, endp***

---

## status

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### Considerations

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the status parameter.

---

**data-set**

|                      |                                                                                                                                                       |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Restriction</b>   | <i>Required.</i> Specifies the name of the required data set.                                                                                         |
| <b>Format</b>        | 4-byte field                                                                                                                                          |
| <b>Consideration</b> | The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the status parameter. |

---

**options**

|                    |                                                                                                             |
|--------------------|-------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Specifies a list of parameters concerned with holding single or multiple records.          |
| <b>Format</b>      | Variable-length field                                                                                       |
| <b>Options</b>     | SECKEY= (xxxxSKyy)<br><i>Required.</i> Specifies the secondary key to be used for the READX function where: |

( ) = Required parentheses

xxxx = Data file name

SK= Invariable, enter as shown

yy = Secondary key identifier

You can shorten the SECKEY keyword to SK.

DIRECTION= FORWARD F or REVERSE R

*Optional.* Instructs the PDM to read secondary keys in either ascending or descending order provided your DBA specified both forward and reverse sorting during secondary key definition. The default DIRECTION value is FORWARD. If the DBA specified only one sort direction, you must specify the same option parameter or the PDM returns an error. If you want to change direction part way through a read, set the function field in the qualifier to REBD. You can shorten the DIRECTION keyword to DR.



FORWARD and REVERSE are always supported regardless of the direction specified in the secondary key definition. REBD is supported for compatibility but is not required to change direction. REBD is equivalent to BEGN.

---

KEYCOUNT= YES Y or NO N

*Optional.* Instructs the PDM to count the number of records in an index file containing the key value specified. When this parameter is set to YES, it causes the PDM to return 4 in the size parameter and the count as a 4-byte binary integer in the data area. No records are retrieved. The default keycount value is NO. You can use this option in conjunction with all other options.

You can shorten the KEYCOUNT keyword to KC.

KEYMATCH= EQUAL EQ, EQUAL NEXT EN, NEXT NX

*Optional.* Specifies key match options for reads, allowing the application to match the full or partial key supplied with:

EQUAL EQ ( equal values only (the default))

EQUAL\_NEXT EN (equal value, or next value)

NEXT NX (next value)

If you supply the partial key, you may omit a character from the right, but you must provide the number of characters specified in the length portion of the qualifier parameter.

With EN, or NX, the next value depends on the sort direction. For keys sorted in forward direction, EN matches values that are greater than or equal to the key supplied and NX matches values that are greater than the key supplied. For keys sorted in reverse direction, EN matches values that are less than or equal to the key supplied, and NX matches values that are less than the key supplied.

You can shorten the KEYMATCH keyword to KM.

**UNIX**

**KEYMATCH= STRING ST** When **STRING ST** (string search) is used, the left-justified key value in the qualifier is used as the search string. The index will be searched in the direction indicated by the **DIRECTION** parameter. Each key found in the index will be searched for the specified string. Only records whose keys contain the string will be returned. As with all **READX** operations, the qualifier will be returned with **NEXT** in the first 4 positions. Subsequent calls with **NEXT** in the qualifier will continue the string search from the last key found in the secondary key index. It is not necessary to use the Mask option with string search. If masking is used, only those characters preceding the first mask character in the key will be used as the string value.

**UNIX**

**MASK= YES Y or NO N** *Optional.* Specifies whether the PDM should use the key value in the qualifier parameter as a literal value or a masked value. The default is **NO**, implying that the key value is a literal value. When set to **YES**, the key value must be a full-length secondary key masked value. See the Qualifier Parameter for “**READX**” on page 314 for the rules of key mask construction.

You can shorten the MASK keyword to MK.

## Considerations

- ◆ You can specify any or all of the option parameters separated by commas and terminated by **END**.
- ◆ You can specify each option more than once, but the PDM uses only the last occurrence.
- ◆ The following is an example:

```
SK= (xxxxSKyy) , DR=R , KM=EN , KC=N , END .
```

**qualifier**

**Description**     *Required.* Specifies parameters concerned with processing serial records.

**Format**            A variable-length field with the following format:  
*qqqqnnnnrrrrrrsssskkkkkk...*

**Options**            The following table lists the parameter values that can be used with READX.

| Value      | Format                  | Meaning                                                                       |
|------------|-------------------------|-------------------------------------------------------------------------------|
| qqqq       | BEGN                    | New read (implicit bind of options).                                          |
|            | NEXT                    | Continue with read.                                                           |
|            | REBD                    | Rebind the qualifier (used for direction change).                             |
|            | END                     | Release all context when passed. End of index has been reached when returned. |
| nnnn       | (none)                  | Reserved for internal use—do not modify.                                      |
|            |                         | For UNIX, this field is the index context id.                                 |
| rrrr       | (4-byte binary integer) | RRN of data record returned.                                                  |
| ssss       | (numeric integer)       | Size of input value for secondary key.                                        |
| kkkkkkk... | (value)                 | Input value for secondary key.                                                |

## Consideration

### UNIX

MASK= YES A full-length *kkkk* can be masked by using MASK=YES in the options parameter and using the masking characters as described below within the *kkkk*. The search is performed by reading sequentially through the index and comparing each key returned to the masked *kkkk* value supplied. If a match is found, the corresponding record in the dataset is read and returned to the application program. If a match is not found, the next sequential key is read from the index. This mechanism can obviously cause extended delays while a search is being performed. Exercise caution in building and using masked *kkkk* values, especially against large datasets. The construction of the mask should specify the minimal search required to achieve your data-retrieval goal.

A mask can consist of any valid ASCII value. The characters &, #, and @ in the value have special meanings to the mask processor. All other characters in the mask mean that the secondary key value must match exactly in those positions to qualify for selection. The comparison is made as if all bytes are character. There are no considerations for binary, packed, zoned, and so on. The following lists the special meaning given to the mask characters:

- ◆ & Any valid ASCII value can be in this position of the secondary key.
- ◆ # A decimal digit 0–9 must be in this position of the secondary key.
- ◆ @ An uppercase or lowercase character must be in this position of the secondary key.

The following are examples:

- ◆ The mask B100####AA will return all records with keys B100 followed by any three numeric digits and ending with AA. A forward search will end when the first record beginning with a character greater than B is encountered. A reverse search will end when the first record beginning with a character less than B is encountered.
- ◆ The mask &100####AA will return all records with any alphanumeric character in the first position followed by 100 followed by any three numeric digits followed by AA. This mask will cause a scan of the entire secondary key index for the dataset.

As with all READX operations, the qualifier will be returned with NEXT in the first 4 positions. Subsequent calls with NEXT in the qualifier will continue the masked search from the last key found in the secondary key index.

**area-length**

|                    |                                                                                                                                                          |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> For the KEYCOUNT function, the PDM returns the value of 4, meaning there are 4 bytes in the data area which contain the KEYCOUNT value. |
| <b>Format</b>      | 4-byte binary integer                                                                                                                                    |

---

**data-item-list**

|                    |                                                                                                                                                                                                                                                                                                                                                                      |
|--------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “Data list parameter keywords” on page 233.) |
| <b>Format</b>      | Variable-length field in the following format:                                                                                                                                                                                                                                                                                                                       |

*dataitem1dataitem2...dataitemnEND.*



## Considerations

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Physical keys (CUSTCTRL)
  - Record codes (ORDRCODE)
  - Special data item processing keywords (\*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See [“Using extended data item processing”](#) on page 369 for information on Extended Data Item Processing.
- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.
- ◆ The following are two examples of data item lists:

```
CUSTCTRLCUSTADDRCUSTCRENDEND.
BINDORDRCUSTORDRPRODEND.
```

## ***data-area***

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

### **Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

---

**endp**

**Description**     *Required.* Points to a field that delimits the parameter list.

**Format**            4-character field

**Options**            END.     Holds the record after it is read

                      RLSE     Releases the record after it is read

**General consideration**

---

Index context is stored in DATBAS for the caller. The index context area is used by the READX DML function to relocate the position in the index from which a NEXT QUALIFIER function is to be performed. The index qualifier value is established following the execution of a READX DML function containing a BEGN QUALIFIER function. Any number of index context areas may be used. A new index context area is allocated each time an initialized QUALIFIER parameter (for example, the index context id field of the QUALIFIER is set to NULL or spaces) is passed to the READX DML function. This allows the calling program to maintain control of the number and use of each context area. The index context id field of the QUALIFIER is used to associate a specific context area with a particular READX DML call. By maintaining multiple QUALIFIER parameters in the application program, it is possible to have an index context for each index in use by the application or even multiple contexts for the same index. All index context areas are deallocated on SINOF.

---

# RESET

Use RESET to use the Task Log File to back out any database updates performed by the current task since the most recent COMMIT. This backs out the effect of the current transaction.

The PDM accesses the before images in the Task Log File area for this program and reapplies them to the database. The PDM then releases all resources held by the issuing task and resets the Task Log File area for this task.



All read-ahead buffers are released.

**RESET, status, area-length, data-area, endp**

## status

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### Considerations

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the *status* parameter.

---

**area-length**

**Description**     *Required.* Specifies to the PDM the length of the data to be returned from the log file.

**Format**            4-byte binary integer

**Considerations**

- ◆ If no data is to be retrieved by RESET, the area-length may be set to zero.
- ◆ This value must be equal to or greater than the area-length specified in the program when the previous COMIT was performed.

---

**data-area**

**Description**     *Required.* An area to receive the data stored in the log file by the previous COMIT function.

**Format**            Variable-length field with a length large enough to receive the data restored.

---

**endp**

**Description**     *Required.* Points to a field that delimits the parameter list.

**Format**            4-character field

**Options**            END.     Delimits the parameter list (same effect as RLSE)

                         RLSE     Delimits the parameter list (same effect as END.)

**General considerations**

- ◆ When you perform a RESET, all records that have been held by this program since the most recent COMIT point are released and may be used by other currently active programs.
- ◆ If a COMIT has not been done since the SINON, the task will be reset to the SINON.
- ◆ If task logging is not active, this command releases any resources held by the issuing task.
- ◆ The area length should be greater than or equal to the maximum length specified in corresponding COMIT commands.

## RQLOC

Use RQLOC (Request Location) to accept a key value as input and generate an RRN as output. RQLOC performs no I/O operations. The RRN is the physical location within the primary file of the record whose key value is in the *physical-key* parameter.

Using this function, you can build a data set of physical keys and their respective RRNs. This data set could be sorted by RRN and the physical keys, then processed at maximum speed against the data set.

---

**RQLOC, *status*, *data-set*, *physical-key*, *data-area*, endp**

---

---

### ***status***

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

#### **Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDML Messages and Codes Reference Manual (PDML/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the *status* parameter.

---

### ***data-set***

**Restriction**     *Required.* Specifies the name of the required data set.

**Format**            4-byte field

**Consideration** The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the *status* parameter.

---

***physical-key***

- Description**     *Required.* Points to a field containing the key of the primary record to be processed. The PDM uses this key to locate a primary record.
- Format**             Variable-length as defined on the Directory
- Consideration**     During the command processing, if the *physical-key* parameter does not match the corresponding field in your data area, a status code informs you of the failure. To avoid this, you should use the *physical-key* field name in the data area for this parameter, rather than define a separate field.

---

***data-area***

- Description**     *Required.* A field that receives an RRN value.
- Format**             4-byte binary integer

---

**endp**

- Description**     *Required.* Points to a field that delimits the parameter list.
- Format**             4-character field
- Options**             END.     Delimits the parameter list (same effect as RLSE)
- RLSE     Delimits the parameter list (same effect as END.)

**General considerations**

- ◆ The specified data set must be a primary data set.
- ◆ The data set need not be opened.
- ◆ The data area should be defined as a 4-byte binary number.
- ◆ The program must be signed on to SUPRA Server.

## SINOF

Use SINOF (Sign off) to close all data sets open to this task and disconnect the program from SUPRA Server. If any before-image records are contained in the task log area for this program, which signify database updates without a subsequent COMMIT, a COMMIT is performed as part of the SINOF processing.

---

**SINOF, *status*, *access-control*, *endp***

---

---

### ***status***

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

### **Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the *status* parameter.



**access-control**

**Description**     *Required.* Defines the options and data sets used for the current program run.

**Format**            Variable-length field containing the following:

| -----REALM----- |               |                           |                |        |                     |                     |        |                     |                     |        |      |
|-----------------|---------------|---------------------------|----------------|--------|---------------------|---------------------|--------|---------------------|---------------------|--------|------|
| FIELD<br>NAME   | PROGM<br>NAME | COMPILED<br>DBASE<br>DESC | ACCESS<br>MODE | FILLER | DATA<br>SET<br>NAME | DATA<br>SET<br>MODE | STATUS | DATA<br>SET<br>NAME | DATA<br>SET<br>MODE | STATUS | END. |
| FIELD<br>LENGTH | 8             | 6                         | 6              | 2      | 4                   | 4                   | 4      | 4                   | 4                   | 4      | 4    |
| EXAMPLE         | PRODWK        | PRODDB                    | UPDATE         | ..     | PROD                | SHRE                | ....   | CUST                | PRIV                | ....   | END. |

**PROGRAM NAME**

The 8-character name of the program.

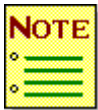
**COMPILED DATABASE DESCRIPTION**

The 6-character name of the compiled database description.

**ACCESS MODE**

The 6-character field identifying the intent of the program. Must be one of the following:

- RDONLY    Only read functions are permitted.
- SINGLE    Single thread the database and turn off logging and record holding. If you use this option, your program should not include reset logic. In addition, if your program runs into problems after updates are made to data, you should restore the database files.



**No other task may sign on while a single-mode task is signed on.**

- UPDATE    All PDML functions are permitted.

**FILLER**

A 2-character field reserved for future use (code \*\*).

**REALM**

A group of 12-character entries consisting of DATA SET NAME, DATA SET MODE, and STATUS—one for each data set in the database required for this program. END. terminates REALM.



---

REALM is optional. If REALM is specified, the files are opened at SINON. If REALM is not specified, the files will be opened dynamically, as specified by any file-oriented PDML command (ADD-M). When files are opened dynamically, the DATA SET MODE will default to SHRE (see below).

---

**DATA SET NAME**

A 4-character field containing a data set name as defined in the SUPRA Server compiled database description.

**DATA SET MODE**

A 4-character field defining the mode of data set sharing for the data set in this entry. Permitted modes are:

- SHRE The data set may be simultaneously read and updated by other programs. This is the default mode.
- PRIV The data set is exclusively assigned to this program; no other program can write to it. Other tasks may read it.
- RDLY This mode is valid only in combination with an access mode of UPDATE. It specifies that the data set is to be accessed for read-only.

**STATUS**

A 4-character field used to return a status for each data set opened. After SINON, this field contains \*\*\*\* if the data set has been opened successfully. If it was not opened successfully, this field contains an error status code indicating the cause of failure (\*MIO, \*SIO, FNTF, IOER, IVTF, LOCK, MODE, or DUPO). For more information on these error statuses, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022.

**END.**

A 4-character field to terminate REALM.

---

**endp**

**Description**     *Required.* Points to a field that delimits the parameter list.

**Format**            4-character field

**Options**            END.     Delimits the parameter list (same effect as RLSE)

                         RLSE     Delimits the parameter list (same effect as END.)

**General considerations**

- ◆ All subsequent commands except SINON return a status of NOSO, indicating you need to SINON. A new SINON can specify the same options or different options from the previous SINON.
- ◆ After SINOF, the data set status field contains \*\*\*\* if the data set has been closed successfully. If the data set has not been closed successfully, an error status code is returned.
- ◆ If any data set status returned in the REALM area is not \*\*\*\*, then the overall status of SINOF is set to the first error status in the REALM. You should then check each data set status to identify which one(s) failed to close.
- ◆ If an I/O error is indicated anywhere in the REALM, then the overall status of SINOF is set to IOER, indicating an I/O error.
- ◆ When task logging is active and the program exits or abends without performing a SINOF, the PDM performs a RESET and SINOF.
- ◆ The content of the access-control parameter should be identical to the access-control parameter used at SINON time. See “**SINON**” on page 332 for a discussion of its format.
- ◆ If a data set specified in the REALM registers an error, SUPRA Server returns the relevant error code in the data set status field within REALM. If a data set not specified in the REALM but opened with a default mode of SHRE registers an error, the details are displayed on the operator’s console.

## SINON

SINON (Signon) must be your program's first call to SUPRA Server. Use SINON to connect your application program to the PDM and to perform internal initialization. It also enables your program to specify which data sets it will access, the mode of access and the type of data set sharing needed.

---

**SINON, *status*, *access-control*, *endp***

---

---

### ***status***

**Description**     *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

#### **Considerations**

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the *status* parameter.

**access-control**

**Description**     *Required.* Defines the options and data sets used for the current program run.

**Format**            Variable-length field containing the following:

| -----REALM----- |               |                           |                |        |                     |                     |        |                     |                     |        |      |
|-----------------|---------------|---------------------------|----------------|--------|---------------------|---------------------|--------|---------------------|---------------------|--------|------|
| FIELD<br>NAME   | PROGM<br>NAME | COMPILED<br>DBASE<br>DESC | ACCESS<br>MODE | FILLER | DATA<br>SET<br>NAME | DATA<br>SET<br>MODE | STATUS | DATA<br>SET<br>NAME | DATA<br>SET<br>MODE | STATUS | END. |
| FIELD<br>LENGTH | 8             | 6                         | 6              | 2      | 4                   | 4                   | 4      | 4                   | 4                   | 4      | 4    |
| EXAMPLE         | PRODWK        | PROddb                    | UPDATE         | ..     | PROD                | SHRE                | ...    | CUST                | PRIV                | ...    | END. |

**PROGRAM NAME**

The 8-character name of the program.

**COMPILED DATABASE DESCRIPTION**

The 6-character name of the compiled database description.

**ACCESS MODE**

The 6-character field identifying the intent of the program. Must be one of the following:

- RDONLY    Only read functions are permitted.
- SINGLE    Single thread the database and turn off logging and record holding. If you use this option, your program should not include reset logic. In addition, if your program runs into problems after updates are made to data, you should restore the database files.



**No other task may sign on while a single-mode task is signed on.**

- UPDATE    All PDML functions are permitted.

**FILLER**

A 2-character field reserved for future use (code \*\*).

**REALM**

A group of 12-character entries consisting of DATA SET NAME, DATA SET MODE, and STATUS—one for each data set in the database required for this program. END. terminates REALM.



---

REALM is optional. If REALM is specified, the files are opened at SINON. If REALM is not specified, the files will be opened dynamically, as specified by any file-oriented PDML command (ADD-M). When files are opened dynamically, the DATA SET MODE will default to SHRE (see below).

---

**DATA SET NAME**

A 4-character field containing a data set name as defined in the SUPRA Server compiled database description.

**DATA SET MODE**

A 4-character field defining the mode of data set sharing for the data set in this entry. Permitted modes are:

- SHRE The data set may be simultaneously read and updated by other programs. This is the default mode.
- PRIV The data set is exclusively assigned to this program; no other program can write to it. Other tasks may read it.
- RDLY This mode is valid only in combination with an access mode of UPDATE. It specifies that the data set is to be accessed for read-only.

**STATUS**

A 4-character field used to return a status for each data set opened. After SINON, this field contains \*\*\*\* if the data set has been opened successfully. If it was not opened successfully, this field contains an error status code indicating the cause of failure (\*MIO, \*SIO, FNTF, IOER, IVTF, LOCK, MODE, or DUPO). For more information on these error statuses, refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022.

**END.**

A 4-character field to terminate REALM.

---

**endp**

|                    |                                                                                                                  |
|--------------------|------------------------------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Points to a field that delimits the parameter list.                                             |
| <b>Format</b>      | 4-character field                                                                                                |
| <b>Options</b>     | END. Delimits the parameter list (same effect as RLSE)<br>RLSE Delimits the parameter list (same effect as END.) |

**General considerations**

- ◆ You can perform another SINON after a SINOF (to change access mode).
- ◆ If any data set status returned in the REALM section is not \*\*\*\*, then the overall status of SINON is set to the first error status in the REALM. You should then check each data set status to identify which one(s) failed to open.
- ◆ If an I/O error is indicated anywhere in the REALM, the overall status of SINON is set to IOER, indicating an I/O error.
- ◆ Never specify the log file names in the REALM parameter of the SINON. SUPRA Server determines if task or function logging is required and automatically opens the log files.
- ◆ See the following table for the permitted uses of SINON access modes. The second following table shows the effects of the various combinations on the issuing task and other tasks.
- ◆ During SINON, SUPRA Server uses the REALM portion of the access-control parameter to open the data sets in the specified mode. If the program then attempts to use a data set that has not been specified in REALM, the PDM automatically opens that data set with a data set mode of SHRE.

This means that it is only necessary to specify the files that require RDLY or PRIV data set modes in the REALM. Existing PDML programs operate without modification.

| Task access mode | Data set access mode | Permitted use                                                                                                                                                                                                                 |
|------------------|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RDONLY           | SHRE                 | This task may only use read-only functions with this data set. The data set may be simultaneously accessed by other tasks.                                                                                                    |
| UPDATE           | SHRE                 | This task may issue any function with this data set. The data set may be simultaneously accessed by other tasks.                                                                                                              |
| RDONLY           | PRIV                 | This task may only use read-only functions with this data set. The data set is exclusively assigned to this task, and no other task may have update access to it. Other tasks may read it.                                    |
| UPDATE           | PRIV                 | This task may issue any standard function with this data set. No other task may have update access to it. Other tasks may read it but not with RDONLY/PRIV.                                                                   |
| UPDATE           | RDLY                 | This task may issue read-only functions to this data set but may update other data sets if SHRE or PRIV is specified for those data sets. A data set with a mode of RDLY can be accessed by another task in UPDATE PRIV mode. |



- ◆ Under normal circumstances with a prefixed database, the logical name CSI\_PREFIX is only translated the first time the PDM is accessed by a client application. When the logical name CSI\_REINIT\_ON\_SINON is defined as TRUE, then the logical name CSI\_PREFIX is translated each time the SINON command is used. Depending on the frequency of the SINON/SINOF commands, this could cause a small performance degradation.



| TASK B      | TASK A               |                    |                        |                      |                      |
|-------------|----------------------|--------------------|------------------------|----------------------|----------------------|
|             | RDONLY SHRE          | RDONLY PRIV        | UPDATE SHRE            | UPDATE PRIV          | UPDATE RDLY          |
| RDONLY SHRE | A: READ<br>B: READ   | A: READ<br>B: READ | A: UPDATE<br>B: READ   | A: UPDATE<br>B: READ | A: READ<br>B: READ   |
| RDONLY PRIV | A: READ<br>B: READ   | A: READ<br>B: READ | A: UPDATE<br>B: LOCK   | A: UPDATE<br>B: LOCK | A: READ<br>B: READ   |
| UPDATE SHRE | A: READ<br>B: UPDATE | A: READ<br>B: LOCK | A: UPDATE<br>B: UPDATE | A: UPDATE<br>B: LOCK | A: READ<br>B: UPDATE |
| UPDATE PRIV | A: READ<br>B: UPDATE | A: READ<br>B: LOCK | A: UPDATE<br>B: LOCK   | A: UPDATE<br>B: LOCK | A: READ<br>B: UPDATE |
| UPDATE RDLY | A: READ<br>B: READ   | A: READ<br>B: READ | A: UPDATE<br>B: READ   | A: UPDATE<br>B: READ | A: READ<br>B: READ   |

Task A signs-on first, and task B signs-on next.

A: READ means task A can read this data set.

A: UPDATE means task A can read and update this data set.

B: READ means task B can read this data set.

B: UPDATE means task B can read and update this data set.

B: LOCK means task B gets a LOCK status on this data set on SINON.

# WRITM

Use WRITM (Write Primary) to update a previously read primary record. SUPRA Server moves the required data items from the data area and rewrites the record.

**WRITM, status,data-set,physical-key,data-item-list,data-area,endp**

## status

**Description**      *Required.* A field into which SUPRA Server places a status code indicating the result of the command.

**Format**            4-character field

## Considerations

- ◆ The value of the status code indicates either the successful completion of the operation (\*\*\*\*) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.
- ◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes. See “[Checking the status parameter](#)” on page 359 for additional information about the *status* parameter.

---

**data-set**

- Restriction**     *Required.* Specifies the name of the required data set.
- Format**           4-byte field
- Consideration** The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the *status* parameter.

---

**physical-key**

- Description**     *Required.* Specifies the physical key of the appropriate record on a primary or related data set.
- Format**           Variable-length field
- Consideration** The PDM determines a Relative Record Number (RRN) for each record by performing a calculation on the physical key's value. This process is the relative address calculator. The PDM uses the RRN to add or read the primary record. The length of the physical key for the relevant primary data set is defined in the compiled database description. When you are performing a related record function, this parameter specifies the key of the controlling primary record.

**data-item-list**

**Description**     *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “[Data list parameter keywords](#)” on page 233.)

**Format**            Variable-length field in the following format:

*dataitem1dataitem2...dataitemnEND.*

**Considerations**

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Special data item processing keywords(\*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.
- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.
- ◆ Data items not coded in the data item list are not changed.
- ◆ The following is an example of a data item list:

*CUSTADDRCUSTCREDEND.*

---

**data-area**

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

**Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

---

**endp**

**Description**     *Required.* Points to a field that delimits the parameter list.

**Format**            4-character field

**Options**            END.     Delimits the parameter list (same effect as RLSE)

                      RLSE     Delimits the parameter list (same effect as END.)

**General considerations**

- ◆ Data items not coded in the data item list are not changed.
- ◆ You cannot modify the physical key using WRITM. To change the physical key, use DEL-M to delete the record, and then use ADD-M to add it with the new key. See “**DEL-M**” on page 276 for information about DEL-M and “**ADD-M**” on page 238 for information about ADD-M.

# WRITV

Use WRITV (Write Related) to update the related record whose RRN is in the reference parameter. SUPRA Server moves the required data items from the data area and rewrites the record.

**WRITV, status, data-set, reference, linkpath, physical-key, data-item-list, data-area, endp**

---

## status

|                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>    | <i>Required.</i> A field into which SUPRA Server places a status code indicating the result of the command.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
| <b>Format</b>         | 4-character field                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Considerations</b> | <ul style="list-style-type: none"><li>◆ The value of the status code indicates either the successful completion of the operation (****) or the nature of the failure. If more than one error occurs, only the first error is reported. Certain commands can make multiple violations. Correction of each of the conditions in turn uncovers the next.</li><li>◆ Be sure to include logic in your program to handle and correct the situation if a command fails or if the status code indicates some special condition other than failure. Refer to the <i>SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX &amp; VMS)</i>, P25-0022, for a complete list of PDML status codes. See “<a href="#">Checking the status parameter</a>” on page 359 for additional information about the <i>status</i> parameter.</li></ul> |

---

## data-set

|                      |                                                                                                                                                              |
|----------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Restriction</b>   | <i>Required.</i> Specifies the name of the required data set.                                                                                                |
| <b>Format</b>        | 4-byte field                                                                                                                                                 |
| <b>Consideration</b> | The data set must be defined in the compiled database description. If the data set is not defined, SUPRA Server returns FNTF in the <i>status</i> parameter. |

---

**reference**

|                      |                                                                                                                                                                       |
|----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Description</b>   | <i>Required.</i> Points to a field identifying the RRN of the specific record to be updated. You place the RRN in this field to tell the PDM which record to process. |
| <b>Format</b>        | 4 alphanumeric characters or a 4-byte binary integer                                                                                                                  |
| <b>Consideration</b> | WRITV does not change the RRN of the record.                                                                                                                          |

---

**linkpath**

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                               |             |                                                        |    |                                  |    |                                                                                               |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------|--------------------------------------------------------|----|----------------------------------|----|-----------------------------------------------------------------------------------------------|
| <b>Description</b> | <i>Required.</i> Specifies the name of the linkpath as defined in the compiled database description. All related record functions use this parameter to indicate which related record list is being processed.                                                                                                                                                                                                                |             |                                                        |    |                                  |    |                                                                                               |
| <b>Format</b>      | 8-byte field with the following format:<br><br><div style="text-align: center;"><i>ppppLKxx</i></div> <p>where:</p> <table> <tr> <td><i>pppp</i></td><td>represents the name of an associated primary data set.</td></tr> <tr> <td>LK</td><td>is a literal (type in as shown).</td></tr> <tr> <td>xx</td><td>represents the last 2 characters of the linkpath name as defined in the database description.</td></tr> </table> | <i>pppp</i> | represents the name of an associated primary data set. | LK | is a literal (type in as shown). | xx | represents the last 2 characters of the linkpath name as defined in the database description. |
| <i>pppp</i>        | represents the name of an associated primary data set.                                                                                                                                                                                                                                                                                                                                                                        |             |                                                        |    |                                  |    |                                                                                               |
| LK                 | is a literal (type in as shown).                                                                                                                                                                                                                                                                                                                                                                                              |             |                                                        |    |                                  |    |                                                                                               |
| xx                 | represents the last 2 characters of the linkpath name as defined in the database description.                                                                                                                                                                                                                                                                                                                                 |             |                                                        |    |                                  |    |                                                                                               |

**Considerations**

- ◆ The primary data set contains the controlling primary record whose key is given in the primary key parameter. LKxx are the same characters used in the reference parameter to identify the beginning of the list. If you specify an invalid linkpath, the PDM returns MLNF, indicating that the linkpath is not defined in the compiled database description.
- ◆ There are two types of linkpaths: primary and secondary. A primary linkpath is the linkpath specified in add commands—ADDVC, ADDVA, ADDVB and ADDVR. Secondary linkpath refers to all other linkpaths defined for a particular record in the compiled database description.
- ◆ To choose the primary linkpath for a record, you need to know the average length of each record list and the frequency of access along each list. Generally, the primary linkpath should be either the longest record list or the most frequently accessed. When using PDML, *always* use the primary linkpath for ordering records.

## ***physical-key***

**Description**     *Required.* Points to a field containing the key of the record in the primary file named by the linkpath parameter.

**Format**           Variable-length as defined on the Directory

### **Considerations**

- ◆ The PDM determines a Relative Record Number (RRN) for each record by performing a calculation on the physical key's value. This process is the relative address calculator. The PDM uses the RRN to add or read the primary record. The length of the physical key for the relevant primary data set is defined in the compiled database description. When you are performing a related record function, this parameter specifies the key of the controlling primary record.
- ◆ This parameter specifies the key of the controlling primary record.



---

## ***data-item-list***

**Description**     *Required.* Points to a field containing a list of data items. This list acts as a map of the layout of the data area. Compose this list using data names (physical fields) defined on the Directory. (You can also use keywords in your data list, or as your entire data list, to perform special functions; see “[Data list parameter keywords](#)” on page 233.)

**Format**            Variable-length field in the following format:  
                          *dataitem1dataitem2...dataitemnEND.*

### **Considerations**

- ◆ Always terminate this parameter with END.
- ◆ A data item must not be repeated in this list.
- ◆ The data item names in the list can include:
  - Data items (PRODNAME)
  - Record codes (ORDRCODE)
  - Special data item processing keywords(\*CODE=xx, \*\*BIND\*\*)
- ◆ The data item names in the list must *not* include:
  - The ROOT field (CUSTROOT)
  - Linkpaths (CUSTLK21)
- ◆ You can list data items in any order. They are processed in the order listed, not necessarily the order within records on the data set. Only the data items listed are processed. For maximum efficiency, list the data item names in the order generated from the compiled database description. See “[Using extended data item processing](#)” on page 369 for information on Extended Data Item Processing.
- ◆ If a data item list contains an invalid data item, the PDM returns ENTF or IVEL. ENTF is returned if the data set part of the name is valid but the data item does not exist. IVEL is returned when the data item is not in the database for the data set concerned or when the list is invalid for the data record being processed.
- ◆ Data items not coded in the data item list are not changed.
- ◆ The following is an example of a data item list:

```
BIND*CODE=AB*ORDRCUSTORDRPRODEND.
```

**data-area**

**Description**     *Required.* An input/output area for the data items named in the data item list.

**Format**            The structure and characteristics of the data area must conform exactly to the Directory definition of the data items (physical fields) in the data list.

**Considerations**

- ◆ The data item list serves as a map of the data area. The structure and characteristics of the data area must conform exactly to the compiled database description definition of the data items named in the data item list.
- ◆ The *data-area* parameter and the *data-item-list* parameter have corresponding fields. The data item list holds names, and the data area holds a value for each of those names. If you name the physical key in the data item list, its value must be the same in both the data area and physical key parameters.

---

**endp**

**Description**     *Required.* Points to a field that delimits the parameter list.

**Format**            4-character field

**Options**            END.     Delimits the parameter list (same effect as RLSE)  
                         RLSE     Delimits the parameter list (same effect as END.)

**General considerations**

- ◆ Data items you do *not* code in the data item list are not changed.
- ◆ You cannot modify physical keys or record codes using WRITV. Use ADDVR to modify these fields.

# 8

## Optimizing your PDML program

This chapter summarizes common programming procedures that will help you make the best use of your PDML program. Some of these procedures are:

- ◆ Linking your application program
- ◆ Using a logical unit of work
- ◆ Processing primary and related data sets
- ◆ Processing data sets serially
- ◆ Testing database programs
- ◆ Processing data lists
- ◆ Reviewing example programs



---

Under VMS, coding RDML and PDML commands in one program can cause database corruption.

---

## Linking your application program

You must link all application program object modules.

---

**VMS**

You must reference the link options file which provides access to the Physical Data Manager (PDM) shareable image. To do this, use the following command in the link options file:

```
SUPRAPDM/SHAREABLE=NOCOPY
```

You can use full file specifications to link modules from different directories. For more information on linking programs with shareable images, refer to your VMS Linker Reference Manual.

---

**UNIX**

You must link your application program with one of three database access programs located in the bin subdirectory of the SUPRA PDM Install directory. Use the following path to locate these programs:

```
/supra1/$SUPRA1_RELEASE_NUMBER/bin
```

The database access programs are:

- ◆ **csidatbas.o** This is a statically-linked, multitask PDM database access program. Your application must be re-linked when a new version of this database access program is installed.
- ◆ **csidatbas.sl** This is a dynamically-linked, multitask PDM database access program. Your application does not need to be relinked when a new version of the database access program is installed. The new csidatbas.sl replaces the old one in the shared library directory.
- ◆ **csibatbas.o** This is a high-performance statically-linked, single-task PDM database access program for batch applications. Your application must be relinked when a new version of this database access program is installed. For more details on single-task PDM, refer to the *SUPRA Server PDM System Administration Guide (UNIX)*, P25-0132.

In addition to linking your application with one of the preceding database access programs, you must link your application with one of the following internal user exit programs (also located in the bin subdirectory of the SUPRA PDM Install directory):

- ◆ **csiintuser.o** Statically-linked internal user exit program.
- ◆ **csiintuser.sl** Dynamically-linked internal user exit program.

The source to the internal and external user exit programs is located in the src subdirectory of the SUPRA PDM Install directory. For details on using and writing internal and external user exits, refer to the *SUPRA Server PDM System Administration Guide (UNIX)*, P25-0132.

---



---

### Example

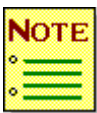
The following UNIX command will compile the C program *progrname.c*, link the object with *csidatbas.sl* and *csiintuser.sl*, and produce the binary executable file *progrname*.

```
cc -o progrname progrname.c /supral/$SUPRA1_RELEASE_NUMBER
/bin/csidatbas.sl/supral/$SUPRA1_RELEASE_NUMBER/bin/csiintuser.sl
```

To compile and link a COBOL program, use the following command:

```
cob -xo progrname progrname.cob -U /supral/$SUPRA1_RELEASE_NUMBER/
bin/csidatbas.sl
/supral/$SUPRA1_RELEASE_NUMBER/bin/csiintuser.sl
```

---



---

It may be necessary to link with the `-ll` option when linking your application with the single-task PDM database access program (*csibatbas.o*).

---

---

## Using logical units of work

A logical unit of work is a group of database requests which must all be completed together. These requests must be concluded by a commit—COMIT command. Using a logical unit of work enables you to enter a complete transaction. SUPRA Server is designed to process transactions or logical units of work.

### Reserving resources

Before your task reserves any resources for updates, all conditions for a logical unit of work must be met. No other task can use these resources until your task frees them at the end of the logical unit of work. To prevent unnecessary resource contention between tasks, limit the time your task holds resources.

## Implementing a logical unit of work

You can implement a logical unit of work by using a standard program structure which includes COMMIT and RESET commands. By using good design techniques, you can avoid unnecessary RESET commands. This will enhance your program efficiency. The following example illustrates a sample program structure for a logical unit of work:

```
INITIALIZATION (SIGN-ON)
 While not finished, do:
 screen input and validation
 Database Update Processing (RDML or PDML)
 If error, then RESET
 else COMMIT
 end
TERMINATION (SIGN-OFF)
```

Once the program is initialized, then data is input from the screen and validated. Initialization (sign on) does the following:

- ◆ Identifies the new task to the system
- ◆ Logically opens the necessary data sets
- ◆ Allocates a unique internal task identifier

---

**VMS**

You can process the database using either RDML or PDML.

---



---

**UNIX**

Process the database using PDML.

---

Your program can either update database records or perform read-only database calls. If your program updates database records, the records are held and cannot be accessed by other programs.

If an error is detected while updating, you can remove the updates by issuing RESET. If no errors are found, issue COMMIT to free any held records. Updated records are now permanent; they may no longer be removed by RESET. During the update process, there should be no further interaction with the user. This process should be as short as possible.

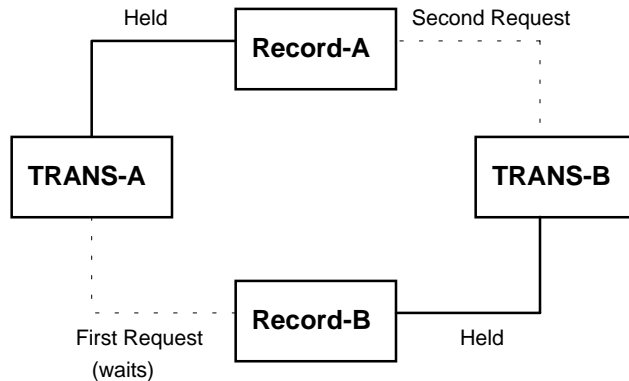
## Understanding deadlocks and how to prevent them

Two transactions might simultaneously request the same database records. This situation is called deadlock, deadly embrace or fatal embrace. To avoid this, all logical units of work should request records in the same sequence. This deadlock can be resolved if one transaction releases all held resources.

Contention between transactions degrades performance because it may be necessary to reexecute functions that cannot complete because the required resource (or records) is being used by another transaction. Therefore, design transactions so the same database records are required by different transactions as infrequently as possible. The following figure illustrates this process.

TRANS-A holds Record-A and starts processing. Meanwhile, TRANS-B holds Record-B and starts processing. At some point TRANS-A tries to hold Record-B. Since it is already held by TRANS-B, TRANS-A waits for Record-B to be released.

During processing, TRANS-B tries to hold Record-A, which is held by TRANS-A. If TRANS-B waits, neither TRANS-A nor TRANS-B will complete since they are waiting on each other.



If you receive a HELD status, indicating the requested record is being held by another transaction, you should do the following:

- ◆ **Wait and retry a few times.** If a HELD status is still received, this might indicate a deadlock.
- ◆ **Reset the transaction.** This backs out or undoes all database requests in the current logical unit of work—since the last commit. This also releases all records held, allowing the deadlocked task to complete.



## Handling errors in a logical unit of work

Errors in a logical unit of work are handled differently, based on where the error was detected:

- ◆ **Detected by the program.** Because the logical unit of work is incomplete when the error is detected, all the processing done before the error must be undone by issuing a RESET. Your program can either start a new logical unit of work or sign off, depending on how severe the error was.
- ◆ **Not detected by the program.** SUPRA Server ensures that each logical unit of work is always complete. Therefore, if task logging is active and SUPRA Server finds that a program is terminating without signing off, the PDM performs a RESET and then signs off.

## Managing your application program

The following sections discuss areas relevant to every application program you will write to access SUPRA Server. These include parameter list definitions, initialization and termination requirements, task management, checking the *status* parameter after the execution of a SUPRA Server command, and recovery guidelines.

### Communicating with SUPRA Server

You communicate with SUPRA Server by using CALL statements. Use standard parameter names when you code your CALL statements. This provides fewer maintenance problems since programs are more easily understood by other programmers. The following are common parameters for each language.

#### Common COBOL parameters

|                  |          |
|------------------|----------|
| U-FUNCTION       | PIC X(5) |
| U-STATUS         | PIC X(4) |
| U-ACCESS-CONTROL | PIC X(n) |
| U-DATA-SET       | PIC X(4) |
| U-REFER          | PIC X(4) |
| U-QUALIFIER      | PIC X(4) |
| U-LINKPATH       | PIC X(8) |
| DDDD-KEY         | PIC X(n) |
| DDDD-LIST        | PIC X(n) |
| DDDD-AREA        | PIC X(n) |
| U-ENDP           | PIC X(4) |

DDDD is the data set name.

The value of *n* depends on the database and the application.

**Common FORTRAN parameters**

|              |                  |
|--------------|------------------|
| CHARACTER*5  | U_FUNCTION       |
| CHARACTER*4  | U_STATUS         |
| CHARACTER*nn | U_ACCESS_CONTROL |
| CHARACTER*4  | U_DATA_SET       |
| CHARACTER*4  | U_REFER          |
| CHARACTER*4  | U_QUALIFER       |
| CHARACTER*8  | U_LINKPATH       |
| CHARACTER*nn | DDDD_KEY         |
| CHARACTER*nn | DDDD_LIST        |
| CHARACTER*nn | DDDD_AREA        |
| CHARACTER*4  | U_ENDP           |

DDDD is the data set name.

The value of  $n$  depends on the database and the application.

**Common BASIC parameters**

|                |                  |
|----------------|------------------|
| DECLARE STRING | U_FUNCTION       |
| DECLARE STRING | U_STATUS         |
| DECLARE STRING | U_ACCESS_CONTROL |
| DECLARE STRING | U_DATA_SET       |
| DECLARE STRING | U_REFER          |
| DECLARE STRING | U_QUALIFER       |
| DECLARE STRING | U_LINKPATH       |
| DECLARE STRING | DDDD_KEY         |
| DECLARE STRING | DDDD_LIST        |
| DECLARE STRING | DDDD_AREA        |
| DECLARE STRING | U_ENDP           |

DDDD is the data set name.

## Parameter list definitions

When you identify the parameters in your CALL statement, follow your installation's conventions. This makes your program more readable and makes debugging easier. For example:

### COBOL

```
01 SINON PIC X(5) VALUE "SINON".
01 READM PIC X(5) VALUE "READM".
01 ENDP PIC X(4) VALUE "END.".
01 STATUS-OK PIC X(4) VALUE "****".
```

### FORTRAN

```
CHARACTER*5 SINON
CHARACTER*5 READM
CHARACTER*4 ENDP
CHARACTER*4 STATUS_OK
DATA SINON/'SINON'/
DATA READM/'READM'/
DATA ENDP/'END.'/
DATA STATUS_OK/'****'/
```

### BASIC

```
DECLARE STRING CONSTANT SINON = "SINON"
DECLARE STRING CONSTANT READM = "READM"
DECLARE STRING CONSTANT ENDP = "END."
DECLARE STRING CONSTANT STATUS_OK = "****"
```

Many installations standardize the data item list and data area parameters that identify the data items to be used. For example:

### COBOL

```
01 CUST-LIST.
 02 FILLER PIC X(8) VALUE "CUSTCTRL".
 02 FILLER PIC X(8) VALUE "CUSTADDR".
 02 FILLER PIC X(4) VALUE "END.".
01 CUST-AREA.
 02 CUST-NAME PIC X(20).
 02 CUST-ADDR PIC X(30).
```

### FORTRAN

```
CHARACTER*20 CUST_LIST
STRUCTURE /CUST_AREA/
 CHARACTER*20 CUST_NAME
 CHARACTER*30 CUST_LIST
END STRUCTURE
DATA CUST_LIST/'CUSTCTRLCUSTADDREND.'/
```

### BASIC

```
RECORD CL
 STRING CL1=8
 STRING CL2=8
 STRING CL3=4
END RECORD CL
DECLARE CL CUST_LIST
RECORD CA
DECLARE CL CUST_LIST
RECORD CA
 STRING CA1=20
 STRING CA2=30
END RECORD CA
DECLARE CA CUST_AREA
CUST_LIST: :CL1="CUSTCTRL"
CUST_LIST: :CL2="CUSTADDR"
CUST_LIST: :CL3="END."
```

This convention controls the data item list and data area definitions in a source statement library. This ensures that all programs in a system refer to the data by the same names. This is invaluable for program maintenance.

## Initialization and termination requirements

To access your SUPRA Server database, your application program must first sign on to SUPRA Server using SINON. The data sets specified in the REALM field of the SINON command are then opened. If your SINON command is unsuccessful (if the status code returned is anything other than \*\*\*\*, \*MIO, \*SIO), no other commands are accepted from that task.

You must open data sets before you use any PDML command referring to those data sets (except RQLOC). If only related data sets are accessed, SUPRA Server requires the associated primary data sets to be opened by the task as well.

To terminate processing of your SUPRA Server database, your application program must issue the SINOF command. This releases all the PDML resources assigned to the program and physically writes all the updated records to disk. The data sets for this task are then logically closed; they are no longer available to this task but might still be open for other tasks.

## Task management

SINON dynamically specifies the SUPRA Server environment for your task to use. It specifies the database, the data sets within that database, and the access and sharing requirements for those data sets. SINOF releases the SUPRA Server resources claimed by the task in its previous SINON to ensure database integrity. Every task that issues a successful SINON to SUPRA Server must do the following before terminating:

- ◆ Issue a SINOF
- ◆ Check the *status* parameter returned from the SINOF

The SUPRA Server exit handler routine performs RESET and SINOF for any task which fails to SINOF before terminating. Your program can change its SUPRA Server environment by issuing a SINOF from its current environment followed by a SINON to the new environment. To maintain efficiency do not change the SUPRA Server environment frequently.

## Checking the *status* parameter

After executing a PDML command, the PDM moves a status code to the *status* parameter to indicate the result of the command. Your application program must check this parameter after every PDML command and take appropriate action. Failing to check the *status* parameter is the most common cause of failure.

When you write a program using SUPRA Server, you should expect certain status codes to be returned. In some cases, successful completion of the operation (a status code of \*\*\*\*) is expected from every CALL to SUPRA Server. In other cases, several status codes are expected, and the program's subsequent processing depends on which status code PDM returns. For example, on a READM command, if \*\*\*\* is returned, the record is available to the user. If the PDM returns MRNF (primary record not found), the record does not exist for the specified physical key value.

In all cases, your program logic must handle all unexpected status codes to ensure they are not ignored. Usually, the appropriate action for all unexpected status codes returned is to produce a formatted report of the PDML parameters that caused the problem.

Refer to the *SUPRA Server PDM Messages and Codes Reference Manual (PDM/RDM Support for UNIX & VMS)*, P25-0022, for a complete list of PDML status codes.

# Using standard primary data-set processing

“Using PDML” on page 227 describes the PDML commands in detail. This section presents a few helpful notes to help you avoid common pitfalls and to increase overall efficiency in primary data set processing.

## Data items you must not refer to

You must not refer to or update the following data items in your data item list. Such a reference could inadvertently change the contents of the file and corrupt your database. If your program refers to these data items, the SUPRA Server PDM produces a status code of IVEL (invalid data item).

|                 |                                                              |
|-----------------|--------------------------------------------------------------|
| <i>ppppROOT</i> | SUPRA Server uses this ROOT data item for internal purposes. |
| <i>ppppLKxx</i> | The PDM automatically updates and maintains the linkpaths.   |



## The ADD-M command

Frequent coding errors occur when using ADD-M because of bad correlation between the physical key, data item list and data area. The *data-area* parameter and the *data-item-list* parameter have corresponding data items. The data item list holds names, and the data area holds a value for each of those names. You must include the record's physical key name in the data item list. Therefore, you must include its value in the corresponding position of the data area. The physical key cannot be blank or binary zeros.

You may define the physical key and *data-area* parameter as the same location within your application program to avoid any problems. This guarantees correlation and ensures that the physical key is always written on the new record. For example:

```
MOVE "ADD-M" TO FUNCTION.
MOVE "P001" TO DATA-SET.
CALL "DATABAS" USING
 FUNCTION
 STAT
 DATA-SET
 P001-CTRL
 P001-DATA-ITEMS
 P001-DATA
 ENDP.
```

where the definitions are:

```
01 P001-DATA-ITEMS
 03 FILLER PIC X(8) VALUE "P001CTRL".
 03 FILLER PIC X(8) VALUE "P001REST".
 03 FILLER PIC X(4) VALUE "END." .

01 P001-DATA.
 03 P001-CTRL PIC X(5) .
 03 P001-REST PIC X(100) .
```

## Structural maintenance during serial processing

You should avoid using add and delete logic on a primary data set while processing it serially with the RDNXT command. Use caution because the application program may not have serial access to certain records after maintenance is performed.

For example, the current record (retrieved serially) might have a synonym which has not yet been read. If the current record is deleted, the PDM automatically optimizes the data set and might move the synonym physically so it is unavailable for serial access.

To avoid this situation, you should perform structural maintenance after the serial processing is complete. This ensures that all records are available for program analysis. The WRITM command executes correctly in this situation since no record movement ever takes place. This also applies if other users are doing updates.

---

## Using standard related-data-set processing

This section presents a few considerations to help you avoid common pitfalls and to increase efficiency in related data set processing.

- ◆ SUPRA Server automatically updates all linkages when you issue an ADD command. The linkpath parameter in the PDML call is used as the first reference point for PDM internal processing and is assumed to be the primary linkpath.
- ◆ Each primary record connected to a related record has a linkpath and a related key. The related key is the physical key of the corresponding primary record. Therefore, to find the key of the primary record to which a related record is connected, it is not necessary to actually read the primary record.
- ◆ Related records for a given data set are all the same length, even though they may have different formats. For efficiency, SUPRA Server does not support variable length related records.
- ◆ If a related record is linked to more than one data set, you must code the related key data item names in the data item list parameter and provide their values in the data area. Otherwise, a BCTL status is returned, indicating a blank or unidentified field specification.

## The reference parameter

PDML uses the reference parameter as a positional indicator for list processing. The reference parameter can have the following values, some provided by your application program and some by SUPRA Server:

- ◆ **LKxx** This literal consists of the last 4 characters of the current linkpath data item name and indicates that the first or last record of a list is to be processed. This value is valid only for READV, READR, ADDVC, ADDVA and ADDVB.
- ◆ **RRN** This is a 4-byte binary number which is the Relative Record Number (RRN). The first record in the data set has an RRN of 1; the third record has an RRN of 3; and so on.
- ◆ **END.** The PDM inserts the literal END. in the reference parameter to indicate that the previous access to this list read the last record of the list. No more records exist on the given linkpath for this physical field. Only SUPRA Server may place END. in the reference parameter. This value must be cleared from reference before another command is called. END. is only returned by SUPRA Server following a READV or a READR. Therefore, after each READV or READR, your program should first check the status field for \*\*\*, and then check for END. in the reference field.

The following table shows the initial setting of the reference parameter when you use the specified command.

| Required action                                      | Command | Set reference                   | Resultant reference                |
|------------------------------------------------------|---------|---------------------------------|------------------------------------|
| Read first record in list                            | READV   | LKxx                            | RRN of the current record          |
| Read next record in list                             | READV   | (Not changed)                   | RRN of the current record          |
| Read next (end of list)                              | READV   | (Not changed)                   | END.                               |
| Read last record in list                             | READR   | LKxx                            | RRN of the current record          |
| Read previous record in list                         | READR   | (Not changed)                   | RRN of the current record          |
| Read previous (end of list)                          | READR   | (Not changed)                   | END.                               |
| Read a record directly                               | READD   | RRN of the desired record       | No change                          |
| Update current record                                | WRITV   | RRN of the current record       | No change                          |
| Add record before current                            | ADDVB   | RRN of the current record       | RRN of the new record              |
| Add record after current                             | ADDVA   | RRN of the current record       | RRN of the new record              |
| Add record at end of list                            | ADDVA   | LKxx                            | RRN of the new record              |
| Add record at end of list                            | ADDVC   | Ignored                         | RRN of the new record              |
| Add record at start of list                          | ADDVB   | LKxx                            | RRN at the new record              |
| Change physical key or record code of current record | ADDVR   | RRN of the current record       | No change                          |
| Delete current record                                | DELVD   | RRN of the current record       | RRN of the previous record<br>LKxx |
| Delete first record in list                          | DELVD   | RRN of the first record in list |                                    |

It is important to repeat the conditions under which SUPRA Server modifies the value of the reference parameter. The following table gives the significant reference value changes after the execution of related PDML commands. The beginning of this section defines the initial setting of the reference parameter.

| Function                     | Initial setting of reference | Final setting of reference                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|------------------------------|------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| READV                        | LKxx                         | <p>The final setting of the reference parameter depends on the status code returned:</p> <ol style="list-style-type: none"> <li>If the status code returned is ****, the reference parameter contains one of the following: <ul style="list-style-type: none"> <li>A. The RRN of the current record in this related data set.</li> <li>B. END. if no record exists for this physical field value.</li> </ul> </li> <li>If a status code other than **** is returned, the reference parameter is not changed.</li> </ol>                                                                             |
| READV                        | RRN                          | <p>The final setting of the reference parameter depends on the status code returned:</p> <ol style="list-style-type: none"> <li>If the status code returned indicates the READV command was successful, the reference parameter contains one of the following: <ul style="list-style-type: none"> <li>A. The RRN of a new record.</li> <li>B. END. if an end-of-list condition has been reached.</li> </ul> </li> <li>If PDM returns a status code other than ****, the reference parameter is not changed.</li> </ol>                                                                              |
| ADDVC,<br>ADDVA, or<br>ADDVB | LKxx or RRN                  | <p>The final setting of the reference parameter depends on the status code returned:</p> <ol style="list-style-type: none"> <li>If the status code returned indicates the ADDVC, ADDVA or ADDVB command was successful, the reference parameter is modified to point to the record just added.</li> <li>If a status code of **** is returned, the reference parameter is not changed.</li> </ol>                                                                                                                                                                                                    |
| DELVD                        | rrrr                         | <p>The final setting of the reference parameter depends on the status code returned:</p> <ol style="list-style-type: none"> <li>If the status code returned indicates that the DELVD command was successful, the reference parameter contains one of the following: <ul style="list-style-type: none"> <li>A. The RRN of the previous record in this list—the record in the list before the one just deleted.</li> <li>B. LKxx if the record just deleted was the first in this list.</li> </ul> </li> <li>If a status code of **** is returned, the reference parameter is not changed.</li> </ol> |

## Improving program efficiency

The following can help you maximize the efficiency of your program:

- ◆ ADDVA, ADDVB, and ADDVC are equally efficient, but keeping lists in a logical sequence (using ADDVA and ADDVB) requires the program to issue additional READVs (or READRs) to get the required position. Therefore, carefully consider the need to have related lists in sequence.
- ◆ The PDM attempts to keep all records within a list physically close together on the disk. Multiple lists are allowed within one related data set. This optimization cannot be maintained for all such lists at the same time. SUPRA Server decides which list it will optimize the storage allocation for. To do this, the PDM always assumes that the linkpath specified in a given ADDVx command (ADDVA, ADDVB and ADDVC) identifies the primary linkpath for that command.

Ensure that the linkpath you code in any ADDV command is always the primary linkpath for the related data set. To optimize retrieval times, when using the READV or READR command, your application program should specify the primary linkpath.

- ◆ Define variable formats in related records (coded related records) for these reasons:
  - To provide a powerful means of storing logically related data items together in a database.
  - To select and connect certain record formats to a given relationship.

To implement coded related records, specify a 2-byte record code for each record in the related data set. When adding a coded record, enter the record code in the user's data area, and enter the *rrr*CODE data item name as a component of the data item list parameter in the PDML CALL.

When reading along a linkpath that connects multiple record types, your program must determine the format of the retrieved records either by verifying the contents of the RRN data item or by using a code directed read (see “[Code-directed reading](#)” on page 370).

- ◆ Use short data item lists. Keep the sequence of the data item list the same as the physical sequence of the data items in the record, whenever possible.
- ◆ Use data item binding to search the data item list only the first time it is used.
- ◆ The PDM modifies the reference parameter as discussed in “[The reference parameter](#)” on page 363. Therefore, you must not modify it unintentionally.

---

## Understanding RDNXT serial processing

Use RDNXT for serial processing. It is an access method for large data sets when the retrieved records are not known uniquely by key, and when only the data actually existing on the data set(s) is to be processed.

Use RDNXT to:

- ◆ Scan a primary data set
- ◆ Set or reset a status field for all, or most, of its records
- ◆ Scan a primary or related data set to produce a data set suitable for report processing
- ◆ Retrieve all the data from the database data sets for backup

When using RDNXT, remember the following:

- ◆ RDNXT operates serially and not sequentially. Therefore, it does not process according to physical key values.
- ◆ RDNXT automatically reads all records, even though it returns control to the program only when the records contain data.
- ◆ RDNXT provides full data independence.

## Testing database programs

Program testing in the database environment is more complex than in a sequential file environment. This is because interrelationships are inherent in a database structure, and the program's actions may affect multiple data sets in any one call statement. Furthermore, there is always a temptation to test the program on the live or production database. This can cause severe problems, even when you use logging.

If you take proper precautions and use simple guidelines, program testing can be simplified. You can also maintain the integrity of a live or production database. To do this, make a backup copy of the test database before you test your program. The production and test databases can be identical except for their names. These names are referenced in SINON and SINOF, and in the file specification for the data sets.

Use the following guidelines for program testing:

- ◆ Whenever possible, construct a mini database for testing purposes. It should have the same logical characteristics as the live database (data set names, database description names, data item names, etc.) but should contain only a subset of the live database's data (about five to ten percent). You should periodically update the test database to reflect the most recent data relationships in the live database.
- ◆ Use the test database to test your programs. After you run your program against the test database, check the program results. If your program has corrupted the database or written erroneous data on its data sets, the test database should be restored to its original condition using the backup copy.



## Using extended data item processing

Data item binding and code directed reading enhance the performance of data item processing while maintaining complete data independence. The general format of the data item list is:

```
dataitem1dataitem2...dataitemnEND.
```

You can supplement the data item notation with either the keywords **\*\*BIND\*\*** or **\*CODE=xx**, or a data item or *sub-data-item* name. The following sections describe these keywords in detail.

### Data-item binding

Data item binding increases the efficiency of SUPRA Server when processing a user data item. It eliminates the database description data item table look-up on all but the initial call to SUPRA Server for a specific data item list. You can use data item binding for any function with a data item list as one of its parameters. To use data item binding, place **\*\*BIND\*\*** in front of the normal data item list. If you are also using code directed reading (see “[Code-directed reading](#)” on page 370), **\*\*BIND\*\*** must precede the first **\*CODE=** statement. SUPRA Server recognizes the characters **\*\*BIND\*\*** on the initial CALL and does the following:

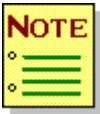
- ◆ Performs the normal database description look-up
- ◆ Replaces the data item list entries
- ◆ Changes the **\*\*BIND\*\*** to **\*BNDxxxx**, where **xxxx** is the identity of the data set being accessed

After these steps complete, your program must not alter the contents of this bound data item list. On subsequent calls to SUPRA Server, this data item list selects the required data items without a full search through the database description table.

## Code-directed reading

Code directed reading is only for related data sets with coded records. It allows you to retrieve selected records from a related data set based on their record codes. Use code directed reading with the READV, READR, READD and RDNXT. It modifies the data item list to incorporate the required record codes. You may select any number of different record codes, but they must occur before any other data items (except **\*\*BIND\*\***).

For each READV or READR issued, the PDM reads along the list of records indicated by the specified linkpath. It returns the first record read containing a code corresponding to one specified in the data item list. Each subsequent use of the command with the same physical key continues from the point in the list where the previous record was read.



---

Specify all record codes at the beginning of the data item list, before any data item definitions. When you request more than one (but not all) record types, specify the redefined data item. Do not use sub-data-items.

---

## PDML examples

You can use SUPRA Server with most programming languages. The examples in this section show you how to implement PDML in COBOL, PL/I and FORTRAN and C. Your Cincom representative can provide further examples.

### COBOL

```

WORKING-STORAGE SECTION.
*
01 CUST-KEY PIC X(6) .
01 CUST-LIST .
 03 FILLER PIC X(8) VALUE "CUSTNAME" .
 03 FILLER PIC X(8) VALUE "CUSTCTYS" .
 03 FILLER PIC X(4) VALUE "END." .
01 CUST-AREA PIC X(80) .
*
01 INPUT-KEY PIC X(6) .
*
01 U-FUNCTION PIC X(5) .
01 U-STATUS PIC X(4) .
01 U-DATA-SET PIC X(4) .
01 U-ENDP PIC X(4) .
*
*
PROCEDURE DIVISION.
A-1.
 MOVE "READM" TO U-FUNCTION.
 MOVE "CUST" TO U-DATA-SET.
 MOVE INPUT-KEY TO CUST-KEY.
 MOVE "RLSE" TO U-ENDP.
 CALL "DATBAS" USING
 U-FUNCTION
 U-STATUS
 U-DATA-SET
 CUST-KEY
 CUST-LIST
 CUST-AREA
 U-ENDP.
 PERFORM U-CHECK-STATUS.
*
*
*
U-CHECK-STATUS.
 .
 .
 .

```

## FORTRAN

```
CHARACTER*4 UENDP, USTAT, UDSET, CUST, RLSE
CHARACTER*5 READM, UFUNC
CHARACTER*6 CKEY
CHARACTER*20 CLIST
CHARACTER*80 CAREA
DATA READM/ 'READM' /
DATA CUST/ 'CUST' /
DATA CLIST/ 'CUSTNAMECUSTCTYSEND.' /
DATA RLSE/ 'RLSE' /
.
.
.
UFUNC=READM
UDSET=CUST
READ(*)CKEY
UENDP=RLSE
CALL DATBAS (UFUNC, USTAT, UDSET, CKEY, CLIST, CAREA, UENDP)
IF (USTAT.NE.'*****') GOTO 900
.
.
.
```

**PL/I**

```

DECLARE DATBAS ENTRY OPTION (FORTRAN);
DECLARE 01 SUPRA_PARAMETERS,
 02 SUPRA_FUNCTION CHARACTER (5),
 02 SUPRA_STATUS CHARACTER (4),
 INITIAL (" "),
 02 SUPRA_DATA_SET CHARACTER (4),
 02 SUPRA_PHYSICAL_KEY CHARACTER (6),
 02 SUPRA_DATA_ITEM_LIST CHARACTER (20),
 02 SUPRA_WORK_AREA CHARACTER (80),
 02 SUPRA_END CHARACTER (4),
 INITIAL ("RLSE");

.
.
.

SUPRA_FUNCTION="READM";
SUPRA_DATA_SET="CUST";
SUPRA_DATA_ITEM_LIST="CUSTNAMECUSTCTYSEND.";
GET LIST (SUPRA_PHYSICAL_KEY);
CALL DATBAS (SUPRA_FUNCTION,
 SUPRA_STATUS,
 SUPRA_DATA_SET,
 SUPRA_PHYSICAL_KEY,
 SUPRA_DATA_ITEM_LIST,
 SUPRA_WORK_AREA,
 SUPRA_ENDP);
IF SUPRA_STATUS "*****" THEN GO TO ERROR;

.
.
.

```

**C**

```
#define SUPRA_FUNCTION "READM"
#define SUPRA_ENDP "RLSE"

char SUPRA_STATUS [4];
char SUPRA_DATA_SET[] = "CUST";
char SUPRA_PHYSICAL_KEY[6];
char SUPRA_DATA_ITEM_LIST[]="CUSTNAMECUSTCTYSAND.";
char SUPRA_WORK_AREA[80];

 DATBAS (SUPRA_FUNCTION,
 SUPRA_STATUS,
 SUPRA_DATA_SET,
 SUPRA_PHYSICAL_KEY,
 SUPRA_DATA_ITEM_LIST,
 SUPRA_WORK_AREA,
 SUPRA_ENDPb/);

if (*(long*)SUPRA_STATUS != *(long*)"*****")
 {
 supra_error(); /* perform error handling routine and
 */
 . /* anything else you want to do when
 */
 . /* you receive an invalid status code
 */
 }
}
```

# A

## Sample RDM programs (VMS)

This appendix provides sample RDM programs demonstrating the use of RDML statements for COBOL, FORTRAN and BASIC. The example programs are not intended to suggest any guidelines or standardization of program structure or coding technique. They are presented in original and expanded versions. The expanded version includes programmer input and SUPRA Server/RDM output.

### Sample COBOL RDM program

#### Input to the preprocessor

```
IDENTIFICATION DIVISION.
 PROGRAM-ID. LVDEMO.
*
*

*
* THIS RELATIONAL DATA MANAGER PROGRAM DEMONSTRATES
* MOST OF THE POSSIBLE RDML EXPANSIONS.
*

*
*
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*
*
01 INCLUDE ULT-CONTROL.
*
01 CUSTOMER INCLUDE CUST (CUST-NO,NAME,CITY).
*
01 INCLUDE CUS-PRD (CUST-NO,PROD-NO).
*
01 CONTACT INCLUDE PROD.
*
01 GENERAL-VARIABLES.
*
```

```

10 USER-ID PIC X(6) VALUE "DEMO94" .
10 USER-PASSWORD PIC X(8) VALUE "DEM-PASS" .
10 PROD-TRAN PIC X(4) .
10 PROD-MARK PIC X(4) .
*
01 INCLUDE PROD.
*
*
PROCEDURE DIVISION.
*
SIGN-ON-DEMO.
*
SIGN-ON USER-ID.
*
SIGN-ON USER-ID USER-PASSWORD.
*
GET-DEMO.
*
GET PROD.
*
GET PROD FOR UPDATE.
*
GET PROD USING PROD-TRAN.
*
GET PROD AT PROD-MARK.
*
GET PROD FOR UPDATE USING PROD-TRAN.
*
GET PROD FOR UPDATE AT PROD-MARK.
*
GET PROD NOT FOUND PERFORM PRODUCT-NOT-FOUND.
*
GET PROD NOT FOUND PERFORM PRODUCT-NOT-FOUND
ELSE PERFORM PRODUCT-ERROR.
*
GET NEXT PROD.
*
GET PRIOR PROD.
*
GET SAME PROD.
*
GET FIRST PROD.
*
GET LAST PROD.
*
INSERT-DEMO.
*
INSERT PROD.
*
INSERT PROD DUP KEY PERFORM DUP-PROD.
*
INSERT NEXT PROD.
*
INSERT PRIOR PROD.

```



```
*
 INSERT LAST PROD.
*
 INSERT FIRST PROD.
*
 DELETE-DEMO.
*
 DELETE PROD.
*
 DELETE ALL PROD.
*
 UPDATE-DEMO.
*
 UPDATE PROD.
*
 MARK-DEMO.
*
 MARK PROD AT PROD-MARK.
*
 RESET-DEMO.
*
 RESET.
*
 RELEASE-DEMO.
*
 RELEASE.
*
 COMMIT-DEMO.
*
 COMMIT.
*
 SIGN-OFF-DEMO.
*
 SIGN-OFF.
*
 STOP RUN.
*
 PRODUCT-NOT-FOUND.
*
 EXIT.
*
 PRODUCT-ERROR.
*
 EXIT.
*
 DUP-PROD.
*
 EXIT.
*
 ERROR-ON-PROD.
*
 EXIT.
```

## Output from the preprocessor

The following is an expanded version of the program statements in the preceding example.

```

IDENTIFICATION DIVISION.
PROGRAM-ID. LVDEMO.
*

*
* THIS RELATIONAL DATA MANAGER PROGRAM DEMONSTRATES
* MOST OF THE POSSIBLE RDML EXPANSIONS.
*

*
* ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
*
*
*01 INCLUDE ULT-CONTROL.
 01 ULT-CONTROL.
 10 ULT-OBJECT-NAME PIC X(30).
 10 ULT-OPERATION.
 20 ULT-ID PIC X(2).
 20 ULT-OPCODE PIC X.
 20 ULT-POSITION PIC X.
 20 ULT-MODE PIC X.
 20 ULT-KEYS PIC X.
 10 ULT-FSI PIC X.
 10 ULT-VSI PIC X.
 10 FILLER PIC X(2).
 10 ULT-MESSAGE PIC X(40).
 10 ULT-PASSWORD PIC X(8).
 10 ULT-OPTIONS PIC X(4).
 10 ULT-CONTEXT PIC X(4).
 10 ULT-LVCONTEXT PIC X(4).
*

```

```

*01 CUSTOMER INCLUDE CUST (CUST-NO,NAME,CITY).
01 LUV-CUSTOMER.
 10 CUSTOMER.
 20 CUST-NO PIC X(006).
 20 NAME PIC X(020).
 20 CITY PIC X(015).
 10 ASI-CUSTOMER.
 20 ASI-CUST-NO PIC X.
 20 ASI-NAME PIC X.
 20 ASI-CITY PIC X.
*
*01 INCLUDE CUS-PRD (CUST-NO,PROD-NO).
01 LUV-CUS-PRD.
 10 CUS-PRD.
 20 CUST-NO PIC X(006).
 20 PROD-NO PIC X(006).
 10 ASI-CUS-PRD.
 20 ASI-CUST-NO PIC X.
 20 ASI-PROD-NO PIC X.
*
*01 CONTACT INCLUDE PROD.
01 LUV-CONTACT.
 10 CONTACT.
 20 PROD-NO PIC X(006).
 20 PROD-DESC PIC X(040).
 20 PROD-RENT PIC S9(07)V9(02) USAGE COMP.
 20 PROD-MAINT PIC S9(07)V9(02) USAGE COMP.
 20 PROD-PURCH PIC S9(07)V9(02) USAGE COMP.
 10 ASI-CONTACT.
 20 ASI-PROD-NO PIC X.
 20 ASI-PROD-DESC PIC X.
 20 ASI-PROD-RENT PIC X.
 20 ASI-PROD-MAINT PIC X.
 20 ASI-PROD-PURCH PIC X.
*
*
01 GENERAL-VARIABLES.
*
 10 USER-ID PIC X(6) VALUE 'DEMO94'.
 10 USER-PASSWORD PIC X(8) VALUE 'DEM-PASS'.
 10 PROD-TRAN PIC X(4).
 10 PROD-MARK PIC X(4).
*

```

```

*
*01 INCLUDE PROD.
01 LUV-PROD.
 10 PROD.
 20 PROD-NO PIC X(006).
 20 PROD-DESC PIC X(040).
 20 PROD-RENT PIC S9(07)V9(02) USAGE COMP.
 20 PROD-MAINT PIC S9(07)V9(02) USAGE COMP.
 20 PROD-PURCH PIC S9(07)V9(02) USAGE COMP.
 10 ASI-PROD.
 20 ASI-PROD-NO PIC X.
 20 ASI-PROD-DESC PIC X.
 20 ASI-PROD-RENT PIC X.
 20 ASI-PROD-MAINT PIC X.
 20 ASI-PROD-PURCH PIC X.
*
*
*
01 ULT-VER-DATA.
 10 ULT-DATE-STAMP.
 20 FILLER PIC X(8) VALUE '19830722'.
 20 FILLER PIC X(6) VALUE '123118'.
 10 ULT-CUSTOMER.
 20 FILLER PIC X(30) VALUE
 'CUST' ' '.
 20 FILLER PIC X(08) VALUE 'CUST-NO, '.
 20 FILLER PIC X(05) VALUE 'NAME, '.
 20 FILLER PIC X(05) VALUE 'CITY, '.
 20 FILLER PIC X(4) VALUE 'END. '.
 10 ULT-CUS-PRD.
 20 FILLER PIC X(30) VALUE
 'CUS-PRD' ' '.
 20 FILLER PIC X(08) VALUE 'CUST-NO, '.
 20 FILLER PIC X(08) VALUE 'PROD-NO, '.
 20 FILLER PIC X(4) VALUE 'END. '.

```

```

10 ULT-CONTACT.
 20 FILLER PIC X(30) VALUE
 'PROD' ' '.
 20 FILLER PIC X(08) VALUE 'PROD-NO, ' '.
 20 FILLER PIC X(10) VALUE 'PROD-DESC, ' '.
 20 FILLER PIC X(10) VALUE 'PROD-RENT, ' '.
 20 FILLER PIC X(11) VALUE
 'PROD-MAINT, ' '.
 20 FILLER PIC X(11) VALUE
 'PROD-PURCH, ' '.
 20 FILLER PIC X(4) VALUE 'END. ' '.
10 ULT-PROD.
 20 FILLER PIC X(30) VALUE
 'PROD' ' '.
 20 FILLER PIC X(08) VALUE 'PROD-NO, ' '.
 20 FILLER PIC X(10) VALUE 'PROD-DESC, ' '.
 20 FILLER PIC X(10) VALUE 'PROD-RENT, ' '.
 20 FILLER PIC X(11) VALUE
 'PROD-MAINT, ' '.
 20 FILLER PIC X(11) VALUE
 'PROD-PURCH, ' '.
 20 FILLER PIC X(4) VALUE 'END. ' '.
PROCEDURE DIVISION.
*
SIGN-ON-DEMO.
*
* SIGN-ON USER-ID.
 MOVE 'LVS---' TO ULT-OPERATION
 MOVE USER-ID
 TO ULT-OBJECT-NAME
 MOVE SPACES TO ULT-PASSWORD
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL
 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-ULT-CONTROL.
*

```

```

* SIGN-ON USER-ID USER-PASSWORD.
 MOVE 'LVS---' TO ULT-OPERATION
 MOVE USER-ID
 TO ULT-OBJECT-NAME
 MOVE USER-PASSWORD
 TO ULT-PASSWORD
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-ULT-CONTROL.
*
GET-DEMO.
*
* GET PROD.
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE 'LVG-RO' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.
*
* GET PROD FOR UPDATE.
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE 'LVG-UO' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.
*

```

```

* GET PROD USING PROD-TRAN.
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE PROD-TRAN
 TO PROD-NO
 OF PROD
 MOVE 'LVG-R1' TO ULT-OPERATION
 CALL 'CSVIPLV' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.
*
* GET PROD AT PROD-MARK,
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE PROD-MARK
 TO ULT-CONTEXT
 MOVE 'LVGAR0' TO ULT-OPERATION
 CALL 'CSVIPLV' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.
*
* GET PROD FOR UPDATE USING PROD-TRAN.
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE PROD-TRAN
 TO PROD-NO
 OF PROD
 MOVE 'LVG-U1' TO ULT-OPERATION
 CALL 'CSVIPLV' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.
*

```

```

* GET PROD FOR UPDATE AT PROD-MARK
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE PROD-MARK
 TO ULT-CONTEXT
 MOVE 'LVGAUO' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.

*
* GET PROD NOT FOUND PERFORM PRODUCT-NOT-FOUND.
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE 'LVG-RO' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*' AND ULT-FSI NOT = 'N'
 PERFORM ERROR-ON-PROD
 ELSE IF ULT-FSI = 'N'
 PERFORM PRODUCT-NOT-FOUND.

* GET PROD NOT FOUND PERFORM PRODUCT-NOT-FOUND
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE 'LVG-RO' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*' AND ULT-FSI NOT = 'N'
 PERFORM ERROR-ON-PROD
 ELSE IF ULT-FSI = 'N'
 PERFORM PRODUCT-NOT-FOUND
 ELSE PERFORM PRODUCT-ERROR.

*

```



```
* GET NEXT PROD.
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE 'LVGNRO' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.

*
* GET PRIOR PROD.
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE 'LVGPRO' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.

*
* GET SAME PROD.
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE 'LVGSRO' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.

*
* GET FIRST PROD.
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE 'LVGFRO' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD. *
```

```

* GET LAST PROD.
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 MOVE 'LVGLRO' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.
*
INSERT-DEMO.
*
* INSERT PROD.
 MOVE 'LVI---' TO ULT-OPERATION
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.
*
* INSERT PROD DUP KEY PERFORM DUP-PROD.
 MOVE 'LVI---' TO ULT-OPERATION
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*' AND ULT-FSI NOT = 'N'
 PERFORM ERROR-ON-PROD
 ELSE IF ULT-FSI = 'N'
 PERFORM DUP-PROD.
*
* INSERT NEXT PROD.
 MOVE 'LVIN--' TO ULT-OPERATION
 MOVE 'PROD' ' TO ULT-OBJECT-NAME
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.

```

```

*
* INSERT PRIOR PROD.
MOVE 'LIVP--' TO ULT-OPERATION
MOVE 'PROD' TO ULT-OBJECT-NAME
CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD. * INSERT LAST PROD.
MOVE 'LVIL--' TO ULT-OPERATION
MOVE 'PROD' TO ULT-OBJECT-NAME
CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.
*
* INSERT FIRST PROD.
MOVE 'LVIF--' TO ULT-OPERATION
MOVE 'PROD' TO ULT-OBJECT-NAME
CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.
*
DELETE-DEMO.
*
* DELETE PROD.
MOVE 'LVD--' TO ULT-OPERATION
MOVE 'PROD' TO ULT-OBJECT-NAME
CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.
*

```

```

* DELETE ALL PROD.
 MOVE 'LVD--*' TO ULT-OPERATION
 MOVE 'PROD' ' ' TO ULT-OBJECT-NAME
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD.

*
UPDATE-DEMO.
*
* UPDATE PROD.
 MOVE 'RDM--' TO ULT-OPERATION
 MOVE 'PROD' ' ' TO ULT-OBJECT-NAME
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 ULT-DATE-STAMP,

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD. *
MARK-DEMO.
*
* MARK PROD AT PROD-MARK.
 MOVE 'LVM--' TO ULT-OPERATION
 MOVE 'PROD' ' ' TO ULT-OBJECT-NAME
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 LUV-PROD,
 LUV-PROD,
 ULT-DATE-STAMP,
 ULT-PROD

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-PROD
 MOVE ULT-CONTEXT TO
 PROD-MARK.

```

```
*
RESET-DEMO.
*
* RESET.
 MOVE 'LVA---' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-ULT-CONTROL,
*
RELEASE-DEMO.
*
* RELEASE.
 MOVE 'LVR---' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-ULT-CONTROL.
*
COMMIT-DEMO.
*
* COMMIT.
 MOVE 'LVC---' TO ULT-OPERATION
 CALL 'CSVIPLVS' USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL

 IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-ULT-CONTROL.
```

```

*
SIGN-OFF-DEMO. *
*
SIGN-OFF.
MOVE 'LVC---' TO ULT-OPERATION
CALL 'CSVIPLVS' USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL

IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-ULT-CONTROL.
MOVE 'LVF---' TO ULT-OPERATION
CALL 'CSVIPLVS' USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL

IF ULT-FSI NOT = '*'
 PERFORM ERROR-ON-ULT-CONTROL.
*
STOP RUN.
*
PRODUCT-NOT-FOUND.
*
EXIT.
*
PRODUCT-ERROR.
*
EXIT.
*
DUP-PROD.
*
EXIT.
*
ERROR-ON-PROD.
*
EXIT.
*
ULT-PROGRAM-END.
GO TO ULT-END-OF-PROGRAM.

ERROR-ON-ULT-CONTROL.
MOVE 'LVA---' TO ULT-OPERATION
CALL 'CSVIPLVS' USING ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL,
 ULT-CONTROL

STOP RUN.
ULT-END-OF-PROGRAM.

&$

```

# Sample FORTRAN RDM program

## Input to the preprocessor

```

 PROGRAM LVDEMO

*
* This FORTRAN Relational Data Manager program demonstrates
* many of the RDML expansions possible.
*

 INCLUDE ULT-CONTROL
 ON ERROR
 TYPE *, 'RDM control call failed', ULT_FSI
 STOP
 END ERROR-HANDLER

*

 INCLUDE PART-COMP=V2(PART=PART-NAME,COMP=COMPONENT-NAME)

*

 INCLUDE PARTS=V1(V1_PART=PART-NAME,FABRICATION-COST)

*

 INCLUDE V2

*

 CHARACTER*6 PART_KEY,COMP_KEY
 CHARACTER*4 PART_MARK
 CHARACTER*8 PASSWORD

*
* SIGN-ON examples.
*

 SIGN-ON 'JADOE'
 SIGN-ON 'JADOE',PASSWORD

```

```
*
* GET examples.
*

 GET PART_COMP
 GET PART_COMP FOR UPDATE
 GET PART_COMP USING PART_KEY
 GET PART_COMP USING PART_KEY,COMP_KEY
 GET PART_COMP AT PART_MARK
 GET PART_COMP FOR UPDATE USING PART_KEY
 GET PART_COMP FOR UPDATE USING PART_KEY,COMP_KEY
 GET PART_COMP FOR UPDATE AT PART_MARK

*

 GET PART_COMP
 NOT FOUND
 GO TO 500
 END IF

*
 GET PART_COMP
 NOT FOUND
 GO TO 500
 ELSE
 TYPE *, 'Record found successfully,'
 END IF

*

 GET NEXT PART_COMP
 GET PRIOR PART_COMP
 GET SAME PART_COMP
 GET FIRST PART_COMP
 GET LAST PART_COMP
```



```
*
* INSERT examples.
*
 INSERT PART_COMP
*
 INSERT PART_COMP
 DUP KEY
 GO TO 500
END IF
*
 INSERT PART_COMP
 DUP KEY
 GO TO 500
ELSE
 TYPE *, 'Record inserted successfully,'
END IF
*
 INSERT NEXT PART_COMP
 INSERT PRIOR PART_COMP
 INSERT FIRST PART_COMP
 INSERT LAST PART_COMP
*
* DELETE example.
*
 DELETE PART_COMP
 DELETE ALL PART_COMP
*
* UPDATE example.
*
 UPDATE PART_COMP
*
* MARK example.
*
 MARK PART_COMP AT PART_MARK
*
* RESET example.
*
 RESET
```

```
*
* RELEASE example.
*
 RELEASE
*
* COMMIT example.
*
 COMMIT
*
* SIGN-OFF example.
*
 SIGN-OFF
*
* Label to which control is passed on record not found.
*
500 STOP
 END
```

## Output from the preprocessor

The following is an expanded version of the program statements in the preceding example.

```

 PROGRAM LVDEMO

 *
 * The purpose of this FORTRAN Relational Data Manager program is to
 * demonstrate many of the RDML expansions possible.
 *
 C INCLUDE ULT-CONTROL
 CHARACTER ULT_OBJECT_NAME*30,ULT_OPERATION*6,ULT_FSI*1,ULT_VSI*1,
+ULT_FILLER*2,ULT_MESSAGE*40,ULT_PASSWORD*8,ULT_OPTIONS*4,
+ULT_CONTEXT*4,ULT_LVCONTEXT*4
 PARAMETER(ULT_CONTROL_LEN=100)
 CHARACTER*(ULT_CONTROL_LEN) ULT_CONTROL
 EQUIVALENCE (ULT_CONTROL(1:30),ULT_OBJECT_NAME(1:30)),
+ (ULT_CONTROL(31:36),ULT_OPERATION(1:6)), (ULT_CONTROL(37:37),
+ULT_FSI(1:1)), (ULT_CONTROL(38:38),ULT_VSI(1:1)),
+ (ULT_CONTROL(39:40),ULT_FILLER(1:2)), (ULT_CONTROL(41:80),
+ULT_MESSAGE(1:40)), (ULT_CONTROL(81:88),ULT_PASSWORD(1:8)),
+ (ULT_CONTROL(89:92),ULT_OPTIONS(1:4)), (ULT_CONTROL(93:96),
+ULT_CONTEXT(1:4)), (ULT_CONTROL(97:100),ULT_LVCONTEXT(1:4))
 CHARACTER*14 ULT_DATE_STAMP
 DATA ULT_DATE_STAMP/'19831114143849'/

 C ON ERROR
 C TYPE *, 'RDM control call failed', ULT_FSI
 C STOP
 C END ERROR-HANDLER
 *

```

```

C INCLUDE PART-COMP=V2 (PART=PART-NAME , COMP=COMPONENT-NAME)
*
 CHARACTER*6 PART
 CHARACTER*6 COMP
 CHARACTER*1 ASI_PART,ASI_COMP
 EQUIVALENCE (PART,PART_COMP(1:6))
 EQUIVALENCE (COMP,PART_COMP(7:12))
 EQUIVALENCE (ASI_PART,PART_COMP(13:13))
 EQUIVALENCE (ASI_COMP,PART_COMP(14:14))
 INTEGER*4 PART_COMP_LEN
 PARAMETER(PART_COMP_LEN=14)
 CHARACTER*(PART_COMP_LEN)PART_COMP
 CHARACTER ULT$PART_COMP*30,ULT$PART*17,ULT$COMP*22,ULT_END_VIEW1*4
 DATA ULT$PART_COMP/'V2' /ULT$PART
+/'006C00PART-NAME','/ULT$COMP/'006C00COMPONENT-NAME','/ULT_END_VIEW
+1/'END.' /
 CHARACTER*73 ULT$1
 EQUIVALENCE (ULT$1,ULT$PART_COMP), (ULT$1(31:47),ULT$PART), (ULT$1
+48:69),ULT$COMP), (ULT$1(70:73),ULT_END_VIEW1)
* C INCLUDE PARTS=V1 (V1_PART=PART-NAME , FABRICATION-COST)
 CHARACTER*6 V1_PART
 INTEGER*4 FABRICATION_COST
 CHARACTER*1 ASI_V1_PART,ASI_FABRICATION_COST
 EQUIVALENCE (V1_PART,PARTS(1:6))
 BYTE ULT_FABRICATION_COST(4)
 EQUIVALENCE (ULB_FABRICATION_COST,FABRICATION_COST), (ULB
+FABRICATION_COST,PARTS(7:10))
 EQUIVALENCE (ASI_V1PART,PARTS(11:11))
 EQUIVALENCE (ASI_FABRICATION_COST,PARTS(12:12))
 INTEGER*4 PARTS_LEN
 PARAMETER(PARTS_LEN=12)
 CHARACTER*(PARTS_LEN)PARTS
 CHARACTER ULT$PARTS*30,ULT$V1_PART*16,ULT$FABRICATION_COST*24,
+ULT_END_VIEW2*4
 DATA ULT$PARTS/'V1' /ULT$V1_PART
+/'006C00PART-NAME','/ULT$FABRICATION_COST
+/'004B00FABRICATION-COST','/ULT_END_VIEW2/'END.' /
 CHARACTER*75 ULT$2
 EQUIVALENCE (ULT$2,ULT$PARTS), (ULT$2(31:47),ULT$V1_PART), (ULT$2
+(48:71),ULT$FABRICATION+COST), (ULT$2(72:75),ULT_END_VIEW2)
*

```

```

C INCLUDE V2
*
 CHARACTER*6 PART_NAME
 CHARACTER*6 COMPONENT_NAME
 INTEGER*4 NO_OF_COMPONENTS
 INTEGER*4 COMPONENT_FABRICATION_COST
 INTEGER*4 COMPONENT_STOCK
 CHARACTER*1 ASI_PART_NAME,ASI_COMPONENT_NAME,ASI_NO_OF_COMPONENTS,
+ASI_COMPONENT_FABRICATION_COST,ASI_COMPONENT_STOCK
 EQUIVALENCE (PART_NAME,V2(1:6))
 EQUIVALENCE (COMPONENT_NAME,V2(7:12))
 BYTE ULB_NO_OF_COMPONENTS(4)
 EQUIVALENCE (ULB_NO_OF_COMPONENTS,NO_OF_COMPONENTS),(ULB
+NO_OF_COMPONENTS,V2(13:16))
 BYTE ULB_COMPONENT_FABRICATION_COST(4)
 EQUIVALENCE (ULB_COMPONENT_FABRICATION_COST,
+COMPONENT_FABRICATION_COST),(ULB_COMPONENT_FABRICATION_COST,V2
+(17:20))
 BYTE ULT_COMPONENT_STOCK(4)
 EQUIVALENCE (ULB_COMPONENT_STOCK,COMPONENT_STOCK),(ULB
+COMPONENT_STOCK,V2(21:24))
 EQUIVALENCE (ASI_PART_NAME,V2(25:25))
 EQUIVALENCE (ASI_COMPONENT_NAME,V2(26:26))
 EQUIVALENCE (ASI_NO_OF_COMPONENTS,V2(27:27))
 EQUIVALENCE (ASI_COMPONENT_FABRICATION_COST,V2(28:28))
 EQUIVALENCE (ASI_COMPONENT_STOCK,V2(29:29))
 INTEGER*4 V2_LEN
 PARAMETER(V2_LEN=29)
 CHARACTER*(V2_LEN)V2
 CHARACTER ULT$V2*30,ULT$PART_NAME*17,ULT$COMPONENT_NAME*22,ULT$
+NO_OF_COMPONENTS*24,ULT$COMPONENT_FABRICATION_COST*34,ULT$
+COMPONENT_STOCK*23,ULT_END_VIEW3*4
 DATA ULT$V2/'V2',ULT$PART_NAME/'/ULT$PART_NAME
+/'006C00PART-NAME','/ULT$COMPONENT_NAME/'006C00COMPONENT-NAME','/
+ULT$NO_OF_COMPONENTS/'004B00NO-OF-COMPONENTS','/ULT$
+COMPONENT_FABRICATION_COST/'004B00COMPONENT-FABRICATION-COST','/
+ULT$COMPONENT_STOCK/'004B00COMPONENT-STOCK','/ULT_END_VIEW3/'END.'/
 CHARACTER*154 ULT$3
 EQUIVALENCE (ULT$3,ULT$V2),(ULT$3(31:47),ULT$PART_NAME),(ULT$3(48:
+69),ULT$COMPONENT_NAME),(ULT$3(70:93),ULT$NO_OF_COMPONENTS),(ULT$3
+(94:127),(ULT$COMPONENT_FABRICATION_COST),(ULT$3(128:150),ULT$
+COMPONENT_STOCK),(ULT$3(151:154),ULT_END_VIEW3)

```

```
*
* SIGN-ON examples.
*
C SIGN-ON 'JADOE'
 ULT_OBJECT_NAME='JADOE'
 ULT_PASSWORD=' '
 ULT_OPERATION='LVS---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 TYPE *, 'RDM control call failed',ULT_FSI
 STOP
 END IF
C SIGN-ON 'JADOE',PASSWORD
*
* GET examples.
*
 ULT_OBJECT_NAME='JADOE'
 ULT_PASSWORD=PASSWORD
 ULT_OPERATION='LVS---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 TYPE *, 'RDM control call failed',ULT_FSI
 STOP
 END IF
C GET PART_COMP
 ULT_OBJECT_NAME='PART_COMP'
 ULT_OPERATION='LVG-RO'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF
C GET PART_COMP FOR UPDATE
 ULT_OBJECT_NAME='PART_COMP'
 ULT_OPERATION='LVG-UO'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF
```

```

C GET PART_COMP USING PART_KEY
 ULT_OBJECT_NAME= ' PART_COMP '
 PART=PART_KEY
 ULT_OPERATION=' LVG-R1 '
 CALL CSVIPLVS(%REF(ULT_CONTROL) , %REF(PART_COMP) ,
+ %REF(ULT_DATE_STAMP) , %REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF
 ULT PART_COMP USING PART_KEY,COMP_KEY
 ULT_OBJECT_NAME= ' PART_COMP '
 PART=PART_KEY
 COMP=COMP_KEY
 ULT_OPERATION=' LVG-R2 '
 CALL CSVIPLVS(%REF(ULT_CONTROL) , %REF(PART_COMP) ,
+ %REF(ULT_DATE_STAMP) , %REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C GET PART_COMP AT PART_MARK
 ULT_OBJECT_NAME= ' PART_COMP '
 ULT_CONTEXT=PART_MARK
 ULT_OPERATION=' LVGAR0 '
 CALL CSVIPLVS(%REF(ULT_CONTROL) , %REF(PART_COMP) ,
+ %REF(ULT_DATE_STAMP) , %REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C GET PART_COMP FOR UPDATE USING PART_KEY
 ULT_OBJECT_NAME= ' PART_COMP '
 PART=PART_KEY
 ULT_OPERATION=' LUG-U1 '
 CALL CSVIPLVS(%REF(ULT_CONTROL) , %REF(PART_COMP) ,
+ %REF(ULT_DATE_STAMP) , %REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

```

```

C GET PART_COMP FOR UPDATE USING PART_KEY,COMP_KEY
 ULT_OBJECT_NAME=' PART_COMP '
 PART=PART_KEY
 COMP=COMP_KEY
 ULT_OPERATION=' LVG-U2 '
 CALL CSVIPLVS(%REF(ULT_CONTROL) ,%REF(PART_COMP) ,
+ %REF(ULT_DATE_STAMP) ,%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C GET PART_COMP FOR UPDATE AT PART_MARK
*
 ULT_OBJECT_NAME=' PART_COMP '
 ULT_CONTEXT=PART_MARK
 ULT_OPERATION=' LVGAU0 '
 CALL CSVIPLVS(%REF(ULT_CONTROL) ,%REF(PART_COMP) ,
+ %REF(ULT_DATE_STAMP) ,%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C GET PART_COMP
 ULT_OBJECT_NAME=' PART_COMP '
 ULT_OPERATION=' LVG-R0 '
 CALL CSVIPLVS(%REF(ULT_CONTROL) ,%REF(PART_COMP) ,
+ %REF(ULT_DATE_STAMP) ,%REF(ULT$PART_COMP))

C NOT FOUND
 IF ((ULT_FSI.NE.'*') .AND. (ULT_FSI.NE.'N')) THEN
 CALL CSVDERROR(ULT_CONTROL)
 ELSE IF (ULT_FSI.EQ.'N') THEN
 GO TO 500
 END IF

*

C GET PART_COMP
 ULT_OBJECT_NAME=' PART_COMP '
 ULT_OPERATION=' LVG-R0 '
 CALL CSVIPLVS(%REF(ULT_CONTROL) ,%REF(PART_COMP) ,
+ %REF(ULT_DATE_STAMP) ,%REF(ULT$PART_COMP))

```



```

C NOT FOUND
 IF ((ULT_FSI.NE.'*') .AND. (ULT_FSI.NE.'N')) THEN
 CALL CSVDERROR(ULT_CONTROL)
 ELSE IF (ULT_FSI.EQ.'N') THEN
 GO TO 500
 ELSE
 TYPE *, 'Record found successfully.'
 END IF

*
C GET NEXT PART_COMP
 ULT_OBJECT_NAME= ' PART_COMP '
 ULT_OPERATION= ' LVGNRO '
 CALL CSVIPLVS(%REF(ULT_CONTROL), %REF(PART_COMP),
+ %REF(ULT_DATE_STAMP), %REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C GET PRIOR PART_COMP
 ULT_OBJECT_NAME= ' PART_COMP '
 ULT_OPERATION= ' LVGPRO '
 CALL CSVIPLVS(%REF(ULT_CONTROL), %REF(PART_COMP),
+ %REF(ULT_DATE_STAMP), %REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C GET SAME PART_COMP
 ULT_OBJECT_NAME= ' PART_COMP '
 ULT_OPERATION= ' LVGSRO '
 CALL CSVIPLVS(%REF(ULT_CONTROL), %REF(PART_COMP),
+ %REF(ULT_DATE_STAMP), %REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

```

```
C GET FIRST PART_COMP
 ULT_OBJECT_NAME=' PART_COMP '
 ULT_OPERATION=' LVGFRO '
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C GET LAST PART_COMP
*
* INSERT examples.
*
 ULT_OBJECT_NAME=' PART_COMP '
 ULT_OPERATION=' LVGLRO '
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C INSERT PART_COMP
*
 ULT_OBJECT_NAME=' PART_COMP '
 ULT_OPERATION=' VLI--- '
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C INSERT PART_COMP
 ULT_OBJECT_NAME=' PART_COMP '
 ULT_OPERATION=' LVI--- '
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
```

```

C DUP KEY
 IF ((ULT_FSI.NE.'*') .AND. (ULT_FSI.NE.'N')) THEN
 CALL CSVDERROR(ULT_CONTROL)
 ELSE IF (ULT_FSI.EQ.'N') THEN
 GO TO 500
 END IF

*
C INSERT PART_COMP
 ULT_OBJECT_NAME=' PART_COMP '
 ULT_OPERATION='LVI--'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))

C DUP KEY
 IF ((ULT_FSI.NE.'*') .AND. (ULT_FSI.NE.'N')) THEN
 CALL CSVDERROR(ULT_CONTROL)
 ELSE IF (ULT_FSI.EQ.'N') THEN
 GO TO 500
 ELSE
 TYPE *, 'Record inserted successfully.'
 END IF

*
C INSERT NEXT PART_COMP
 ULT_OBJECT_NAME=' PART_COMP '
 ULT_OPERATION='LVIN--'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C INSERT PRIOR PART_COMP
 ULT_OBJECT_NAME=' PART_COMP '
 ULT_OPERATION='LVIP--'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

```

```
C INSERT FIRST PART_COMP
 ULT_OBJECT_NAME='PART_COMP'
 ULT_OPERATION='LVIF---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C INSERT LAST PART_COMP
*
* DELETE example.
*
 ULT_OBJECT_NAME='PART_COMP'
 ULT_OPERATION='LVIL--'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C DELETE PART_COMP
 ULT_OBJECT_NAME='PART_COMP'
 ULT_OPERATION='LVD---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF

C DELETE ALL PART_COMP
*
* UPDATE example.
*
 ULT_OBJECT_NAME='PART_COMP'
 ULT_OPERATION='LVD--*'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF
```

```

C UPDATE PART_COMP
*
* MARK example.
*
 ULT_OBJECT_NAME='PART_COMP'
 ULT_OPERATION='LVU---'
 CALL CSVIPVLS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 END IF
C MARK PART_COMP AT PART_MARK
*
* RESET example.
*
 ULT_OBJECT_NAME='PART_COMP'
 ULT_CONTEXT=PART_MARK
 ULT_OPERATION='LVM---'
 CALL CSVIPVLS(%REF(ULT_CONTROL),%REF(PART_COMP),
+ %REF(ULT_DATE_STAMP),%REF(ULT$PART_COMP))
 IF (ULT_FSI.NE.'*') THEN
 CALL CSVDERROR(ULT_CONTROL)
 ELSE
 PART_MARK=ULT_CONTEXT
 END IF
C RESET
*
* RELEASE example.
*
 ULT_OPERATION='LVA---'
 CALL CSVIPVLS(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 TYPE *, 'RDM control call failed',ULT_FSI
 STOP
 END IF
C RELEASE
*

```

```

* COMMIT example.
*
 ULT_OPERATION='LVR---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 TYPE *, 'RDM control call failed',ULT_FSI
 STOP
 END IF
C COMMIT
*
* SIGN-OFF example.
*
 ULT_OPERATION='LVC---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 TYPE *, 'RDM control call failed',ULT_FSI
 STOP
 END IF
C SIGN-OFF
*
* label to which control is passed on record not found.
*
 ULT_OPERATION='LVC---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),,%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 TYPE *, 'RDM control call failed',ULT_FSI
 STOP
 END IF
 ULT_OPERATION='LVF---'
 CALL CSVIPLVS(%REF(ULT_CONTROL),%REF(ULT_CONTROL),
+ %REF(ULT_CONTROL),%REF(ULT_CONTROL))
 IF (ULT_FSI.NE.'*') THEN
 TYPE *, 'RDM control call failed',ULT_FSI
 STOP
 END IF
500 STOP
 END
%$

```

# Sample BASIC RDM program

## Input to the preprocessor

```

100 RDM PROGRAM YYY
 RDM INCLUDE ULT-CONTROL
 RDM ON ERROR
 GOTO 999
 END ERROR-HANDLER
 RDM INCLUDE TEST1
 RDM INCLUDE TEST6G
 RDM ON ERROR
 PRINT ULT_CONTROL::ULT_FSI / GOTO 999
 END ERROR-HANDLER
 RDM INCLUDE TEST6F
 RDM INCLUDE TEST6E
 RDM INCLUDE TEST6D
 RDM INCLUDE TEST7
 RDM INCLUDE FRED=TEST6D(F1=CUST-ORDER-NO,F2=CUST-ORDER-DATE)

200 DO_SIGN_ONS: RDM SIGN-ON 'JADOE'
 RDM SIGN-OFF
 RDM SIGN-ON 'JADOE','JBDOE'

300 DO_GETS: RDM GET TEST1
 RDM GET FIRST TEST1
 RDM GET NEXT TEST1
 RDM GET LAST TEST1
 RDM GET PRIOR TEST1
 RDM GET TEST1 AT TEST1_MARK

!The comments on the next lines test the operation of commenting out
400 DO_GET_USINGS: RDM GET TEST1 USING "PART2" !"!!! + CHR$(52)
 RDM GET FIRST TEST1 USING "!" + CHR$(52)
 RDM GET NEXT TEST1 USING "PART2" + CHR$(52)
 RDM GET LAST TEST1 USING "PART2" + CHR$(52)
 RDM GET PRIOR TEST1 USING "PART2" + CHR$(52)

for i%=1% to 10% !a comment on a normal program line
 print i% !more comments

```

```
500 DO_GET_NOT_FOUND: RDM GET TEST1 USING 'PART2' + CHR$(52)
 NOT FOUND / GOTO 999 / END IF
 RDM GET FIRST TEST1 USING "PART2" + CHR$(52)
 NOT FOUND / GOTO 999 / END IF
 RDM GET NEXT TEST1 USING "PART2" + CHR$(52)
 NOT FOUND / GOTO 999 / END IF(ep)
 RDM GET LAST TEST1 USING "PART2" + CHR$(52)
 NOT FOUND / GOTO 999 / END IF
 RDM GET PRIOR TEST1 USING "PART2" + CHR$(52)
 NOT FOUND / GOTO 999 / END IF
 RDM GET TEST1 AT TEST1 _MARK$NOT FOUND GOTO 999E END IF

600 DO_INSERTS: RDM INSERT TEST6G
 RDM INSERT FIRST TEST6G
 RDM INSERT NEXT TEST6G
 RDM INSERT LAST TEST6G
 RDM INSERT PRIOR TEST6G

700 DO_INSERT_DUPS: RDM INSERT TEST6G / DUP KEY / GOTO 999 / END IF
 RDM INSERT FIRST TEST6G / DUP KEY / GOTO 999 / END IF
 RDM INSERT NEXT TEST6G / DUP KEY / GOTO 999 / END IF
 RDM INSERT LAST TEST6G / DUP KEY / GOTO 999 / END IF
 RDM INSERT PRIOR TEST6G / DUP KEY / GOTO 999 / END IF

800 DO_UPDATE RDM UPDATE TEST 6D
900 DO_DELETES: RDM DELETE TEST6E
 RDM DELETE ALL TEST 6F

950 DO_MISCELLANEOUS: RDM RELEASE
 RDM MARK TEST1 AT TEST1_MARK
 RDM COMMIT
 RDM RESET

999 END
```



## Output from the preprocessor

The following is an expanded version of the program statements in the preceding example.

```

100
!100 RDM PROGRAM YYY
 RDM INCLUDE ULT-CONTROL
RECORD ULT_CONTROL_REC
STRING ULT_OBJECT_NAME = 30, ULT_OPERATION = 6,
ULT_FSI = 1, ULT_VSI = 1, ULT_FILLER = 2, ULT_MESSAGE = 40,
ULT_PASSWORD = 8, ULT_OPTIONS = 4, ULT_CONTEXT = 4, ULT_LVCONTEXT = 4
END RECORD
DECLARE ULT_CONTROL_REC_CONTROL ULT_CONTROL
EXTERNAL SUB CSVIPLVS, CSVBERROR
DECLARE STRING CONSTANT ULT_DATE_STAMP = '19840830161953'
! RDM ON ERROR
!
! GOTO 999
! END ERROR-HANDLER
! RDM INCLUDE TEST1
RECORD TEST1_REC
STRING PART_NAME=6, DECIMAL(4,4) PART_LABOUR_COST, STRING
PART_LAST_UPDATE=6
STRING ASI_PART_NAME = 1, ASI_PART_LABOUR_COST = 1,
ASI_PART_LAST_UPDATE = 1
END RECORD
DECLARE TEST1_REC TEST1
DECLARE STRING CONSTANT TEST1_ULT = 'TEST1'
'006C00PART_NAME, '+'004P04PART-LABOUR-COST, '+'
'006C00PART-LAST-UPDATE, '+'END.'
! RDM INCLUDE TEST6G RECORD TEST6G_REC
STRING CUST=6, STRING ITEM_CORD=6, STRING COMPLETION_DATE=7, STRING
INVOICE_DATE=7, STRING INVOICE_NO_2=6, STRING ITEM_DATE=7, STRING
ITEM_PART=6, STRING DESPATCH_NO=6, STRING INVOICE_NO=6
STRING ASI_CUST = 1, ASI_ITEM_CORD = 1, ASI_COMPLETION_DATE = 1,
ASI_INVOICE_DATE = 1, ASI_INVOICE_NO_2 = 1,
ASI_ITEM_DATE = 1, ASI_ITEM_PART = 1, ASI_DESPATCH_NO = 1,
ASI_INVOICE_NO = 1
END RECORD

```

```

DECLARE TEST6G_REC TEST6G
DECLARE STRING CONSTANT TEST6G_ULT = 'TEST6G'
'006C00CUST,'+'006C00ITEM-CORD,'+'007C00COMPLETION-DATE,'+
'007C00INVOICE-DATE,'+'006C00INVOICE-NO-2,'+'007C00ITEM-DATE,'+
'006C00ITEM-PART,'+'006C00DESPATCH-NO,'+'006C00INVOICE-NO,'+'END.'
! RDM ON ERROR
! PRINT ULT_CONTROL::ULT_FSI
! GOTO 999
! END ERROR-HANDLER
! RDM INCLUDE TEST6F
RECORD TEST6F_REC
STRING CUST_ORDER_NO=6,STRING ITEM_DATE=7,STRING ITEM_PART=6,STRING
DESPATCH_NO=6,STRING INVOICE_NO=6,STRING ITEM_CUST=6,STRING
COMPLETION_DATE=7
STRING ASI_CUST_ORDER_NO = 1,ASI_ITEM_DATE = 1,
ASI_ITEM_PART = 1,ASI_DESPATCH_NO = 1,ASI_INVOICE_NO = 1,
ASI_ITEM_CUST = 1,ASI_COMPLETION_DATE = 1
END RECORD
DECLARE TEST6F_REC TEST6F
DECLARE STRING CONSTANT TEST6F_ULT = 'TEST6F'
'006C00CUST-ORDER-NO,'+'007C00ITEM-DATE,'+'006C00ITEM-PART,'+
'006C00DESPATCH-NO,'+'006C00INVOICE-NO,'+'006C00ITEM-CUST,'+
'007C00COMPLETION-DATE,'+'END.'
! RDM INCLUDE TEST 6E
RECORD TEST6E_REC
STRING CUST_ORDER_NO=6,STRING CUST_ORDER_DATE=9,STRING ITEM_CUST=6,
STRING COMPLETION_DATE=7,STRING ITEM_DATE=7,STRING ITEM_PART=6,STRING
DESPATCH_NO=6,STRING INVOICE_NO=6
STRING ASI_CUST_ORDER_NO = 1,ASI_CUST_ORDER_DATE = 1,
ASI_ITEM_CUST = 1,ASI_COMPLETION_DATE = 1,
ASI_ITEM_DATE = 1,ASI_ITEM_PART = 1,ASI_DESPATCH_NO = 1,
ASI_INVOICE_NO = 1
END RECORD
DECLARE TEST6E_REC TEST6E
DECLARE STRING CONSTANT TEST6E_ULT = 'TEST6E'
006C00CUST-ORDER-NO,'+'009C00CUST-ORDER-DATE,'+'006C00ITEM-CUST,'+
007C00COMPLETION-DATE,'+'007C00ITEM-DATE,'+'006C00ITEM-PART,'+
006C00DESPATCH-NO,'+'006C00INVOICE-NO,'+'END.'
! RDM INCLUDE TEST6D

```

```

RECORD TEST6D_REC
STRING CUST_ORDER_NO=6,STRING CUST_ORDER_DATE=9,STRING ITEM_CUST=6,
STRING COMPLETION_DATE=7,STRING DESPATCH_DATE=7,STRING DESPATCH_NO_1=6,
DECIMAL(4,0) QUANTITY_DESPATCHED
STRING ASI_CUST_ORDER_NO = 1,ASI_CUST_ORDER_DATE = 1,
STRING ASI_CUST_ORDER_NO = 1,ASI_CUST_ORDER_DATE = 1,
ASI_ITEM_CUST = 1,ASI_COMPLETION_DATE = 1,
ASI_QUANTITY_DESPATCHED = 1
END RECORD
DECLARE TEST6D_REC TEST6D
DECLARE STRING CONSTANT TEST6D_ULT = 'TEST6D ' +
'006C00CUST-ORDER-NO, '+'009C00CUST-ORDER-DATE, '+'006C00ITEM-CUST, '+'
'007C00COMPLETION-DATE, '+'007C00DESPATCH-DATE, '+'
'006C00DESPATCH-NO-1, '+'004P00QUANTITY_DESPATCHED, '+'END.'
! RDM INCLUDE TEST7
RECORD TEST7_REC
STRING STOCK_ISSUE_NUMBER=5,STRING STCK_PART=6,STRING TRANSACTION_DATE=
7,STRING AUTHORISATION=11
STRING ASI_STOCK_ISSUE_NUMBER = 1,ASI_STCK_PART = 1,
ASI_TRANSACTION_DATE = 1,ASI_AUTHORISATION = 1
END RECORD
DECLARE TEST7_REC TEST7
DECLARE STRING CONSTANT TEST7ULT = 'TEST7 ' +
005C00STOCK-ISSUE-NUMBER, '+'006C00STCK-PART, '+'
007C00TRANSACTION-DATE, '+'011C00AUTHORISATION, '+'END.'
! RDM INCLUDE FRED=TEST6D(F1=CUST-ORDER-NO,F2=CUST-ORDER-DATE)
RECORD FRED_REC
STRING F1=6,STRING F2=9
STRING ASI_F1 = 1,ASI_F2 = 1
END RECORD
DECLARE FRED_REC FRED
DECLARE STRING CONSTANT FRED_ULT = 'TEST6D ' +
'006C00CUST-ORDER-NO, '+'009C00CUST-ORDER-DATE, '+'END.'
200 DO_SIGN_ONS:
!200 DO_SIGN_ONS: RDM SIGN-ON 'JADOE'
ULT_CONTROL::ULT_OBJECT_NAME='JADOE'
ULT_CONTROL::ULT_PASSWORD=' '
ULT_CONTROL::ULT_OPERATION='LVS---'

```

```

CALL CSVIPLVS(ULT_CONTROL BY REF,ULT_CONTROL BY REF,ULT_CONTROL BY REF,
ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 GOTO 999
END IF
! RDM SIGN-OFF
ULT_CONTROL::ULT_OPERATION='LVC---'
CALL CSVIPLVS(ULT_CONTROL BY REF,ULT_CONTROL BY REF, ULT_CONTROL BY REF,
ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 GOTO 999
END IF
ULT_CONTROL::ULT_OPERATION='LVF---'
CALL CSVIPLVS(ULT_CONTROL BY REF,ULT_CONTROL BY REF,ULT_CONTROL BY REF,
ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 GOTO 999
END IF
! RDM SIGN-ON 'JADOE','JBDOE'
ULT_CONTROL::ULT_OBJECT_NAME='JADOE'
ULT_CONTROL::ULT_PASSWORD='JBDOE'
ULT_CONTROL::ULT_OPERATION='LVS---'
CALL CSVIPLVS(ULT_CONTROL BY REF,ULT_CONTROL BY REF,ULT_CONTROL BY REF,
ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 GOTO 999
END IF

300 DO_GETS:
!300 DO_GETS: RDM GET TEST1

ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
ULT_CONTROL::ULT_OPERATION='LVG-RO'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
CALL CSVBERROR(ULT_CONTROL)
END IF

```

```

! RDM GET FIRST TEST1
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
ULT_CONTROL::ULT_OPERATION='LVGFRO'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
CALL CSVBERROR(ULT_CONTROL)
END IF

! RDM GET NEXT TEST1
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
ULT_CONTROL::ULT_OPERATION='LVGNRO'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
CALL CSVBERROR(ULT_CONTROL)
END IF

! RDM GET LAST TEST1
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
ULT_CONTROL::ULT_OPERATION='LVGLRO'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
CALL CSVBERROR(ULT_CONTROL)
END IF

! RDM GET PRIOR TEST1
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
ULT_CONTROL::ULT_OPERATION='LVGPRO'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
CALL CSVBERROR(ULT_CONTROL)
END IF

! RDM GET TEST1 AT TEST1_MARK
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
ULT_CONTROL::ULT_CONTEXT=TEST1_MARK
ULT_CONTROL::ULT_OPERATION='LVGARO'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)

```

```
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
CALL CSVBERROR(ULT_CONTROL)
END IF

!The comments on the next lines test the operation of commenting out
 400 DO_GET_USINGS:
!400 DO_GET_USINGS: RDM GET TEST1 USING 'PART2' !!'!!!!!!' + CHR$(52)
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
TEST1::PART_NAME='PART2'+CHR$(52)
ULT_CONTROL::ULT_OPERATION='LVG-R1'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
CALL CSVBERROR(ULT_CONTROL)
END IF

! RDM GET FIRST TEST1 USING '!!' + CHR$(52) !!a comment
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
TEST1::PART_NAME='!' +CHR$(52)
ULT_CONTROL::ULT_OPERATION='LVGFR1'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF, TEST1_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
END IF

! RDM GET NEXT TEST1 USING 'PART2' + CHR$(52)
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
TEST1::PART_NAME='PART2'+CHR$(52)
ULT_CONTROL::ULT_OPERATION='LVGNR1'
CALL CSVIPLVS(ULT_CONTROL BY REF,,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
CALL CSVBERROR(ULT_CONTROL)
END IF

! RDM GET LAST TEST1 USING 'PART2' + CHR$(52)
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
TEST1::PART_NAME='PART2'+CHR$(52)
ULT_CONTROL::ULT_OPERATION='LVGLR1'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
```

```

IF (ULT_CONTROL::ULT_FSI <> '*') THEN
CALL CSVBERROR(ULT_CONTROL)
END IF

!RDM GET PRIOR TEST1 USING 'PART2' + CHR$(52)
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
TEST1::PART_NAME='PART2'+CHR$(52)
ULT_CONTROL::ULT_OPERATION='LVGPR1'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
CALL CSVBERROR(ULT_CONTROL)
END IF

 for i%=1% to 10% !a comment on a normal program line
 print i% !more comments
 next i%

 500 DO_GET_NOT_FOUND:
!500 DO_GET_NOT_FOUND: RDM GET TEST1 USING 'PART2' +CHR$(52)
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
TEST1::PART_NAME='PART2'+CHR$(52)
ULT_CONTROL::ULT_OPERATION='LVG-R1'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
! NOT FOUND
IF ((ULT_CONTROL::ULT_FSI <> '*') AND (ULT_CONTROL::ULT_FSI <> 'N')) THEN
CALL CSVBERROR(ULT_CONTROL)
END IF
IF (ULT_CONTROL::ULT_FSI='N') THEN
GOTO 999
END IF

! RDM GET FIRST TEST1 USING 'PART2' + CHR$(52)
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
TEST1::PART_NAME='PART2'+CHR$(52)
ULT_CONTROL::ULT_OPERATION='LVGFR1'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
! NOT FOUND

```

```
IF ((ULT_CONTROL::ULT_FSI <> '*') AND (ULT_CONTROL::ULT_FSI <> 'N')) THEN
CALL CSVBERROR(ULT_CONTROL)
END IF
IF (ULT_CONTROL::ULT_FSI='N') THEN
GOTO 999
END IF
! RDM GET NEXT TEST1 USING 'PART2' + CHR$(52)
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
TEST1::PART_NAME='PART2'+CHR$(52)
ULT_CONTROL::ULT_OPERATION='LVGNR1'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
! NOT FOUND
IF ((ULT_CONTROL::ULT_FSI <> '*') AND (ULT_CONTROL::ULT_FSI <> 'N')) THEN
CALL CSVBERROR(ULT_CONTROL)
END IF
IF (ULT_CONTROL::ULT_FSI='N') THEN
GOTO 999
END IF
! RDM GET LAST TEST1 USING 'PART2' + CHR$(52)
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
TEST1::PART_NAME='PART2'+CHR$(52)
ULT_CONTROL::ULT_OPERATION='LVGLR1'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
! NOT FOUND
IF ((ULT_CONTROL::ULT_FSI <> '*') AND (ULT_CONTROL::ULT_FSI <> 'N')) THEN
CALL CSVBERROR(ULT_CONTROL)
END IF
IF (ULT_CONTROL::ULT_FSI='N') THEN
GOTO 999
END IF
! RDM GET PRIOR TEST1 USING 'PART2' + CHR$(52)
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
TEST1::PART_NAME='PART2'+CHR$(52)
ULT_CONTROL::ULT_OPERATION='LVGPR1'
```



```

CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
! NOT FOUND
IF ((ULT_CONTROL::ULT_FSI <> '*') AND (ULT_CONTROL::ULT_FSI <> 'N')) THEN
CALL CSVBERROR(ULT_CONTROL)
END IF
IF (ULT_CONTROL::ULT_FSI='N') THEN
GOTO 999
END IF
! RDM GET TEST1 AT TEST1_MARK
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
ULT_CONTROL::ULT_CONTEXT=TEST1_MARK
ULT_CONTROL::ULT_OPERATION='LVGARO'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
! NOT FOUND
IF ((ULT_CONTROL::ULT_FSI <> '*') AND (ULT_CONTROL::ULT_FSI <> 'N')) THEN
CALL CSVBERROR(ULT_CONTROL)
END IF
IF (ULT_CONTROL::ULT_FSI='N') THEN
GOTO 999
END IF

 600 DO_INSERTS:
!600 DO_INSERTS: RDM INSERT TEST6G
ULT_CONTROL::ULT_OBJECT_NAME='TEST6G'
ULT_CONTROL::ULT_OPERATION='LVI-U-'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6G BY REF,
ULT_DATE_STAMP BY REF,TEST6G_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 PRINT ULT_CONTROL::ULT_FSI
 GOTO 999
END IF
! RDM INSERT FIRST TEST6G
ULT_CONTROL::ULT_OBJECT_NAME='TEST6G'
ULT_CONTROL::ULT_OPERATION='LVIFU-'

```

```

CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6G BY REF,
ULT_DATE_STAMP BY REF,TEST6G_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 PRINT ULT_CONTROL::ULT_FSI
 GOTO 999
END IF
! RDM INSERT NEXT TEST6G
ULT_CONTROL::ULT_OBJECT_NAME='TEST6G'
ULT_CONTROL::ULT_OPERATION='LVINU-'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6G BY REF,
ULT_DATE_STAMP BY REF,TEST6G_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 PRINT ULT_CONTROL::ULT_FSI
 GOTO 999
END IF
! RDM INSERT LAST TEST6G
ULT_CONTROL::ULT_OBJECT_NAME='TEST6G'
ULT_CONTROL::ULT_OPERATION='LVILU-'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6G BY REF,
ULT_DATE_STAMP BY REF,TEST6G_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 PRINT ULT_CONTROL::ULT_FSI
 GOTO 999
END IF
! RDM INSERT PRIOR TEST6G
ULT_CONTROL::ULT_OBJECT_NAME='TEST6G'
ULT_CONTROL::ULT_OPERATION='LVIPU-'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6G BY REF,
ULT_DATE_STAMP BY REF,TEST6G_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 PRINT ULT_CONTROL::ULT_FSI
 GOTO 999
END IF

```

```

700 DO_INSERT_DUPS:
!700 DO_INSERT_DUPS: RDM INSERT TEST6G
ULT_CONTROL::ULT_OBJECT_NAME='TEST6G'
ULT_CONTROL::ULT_OPERATION='LVI-U-'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6G BY REF,
ULT_DATE_STAMP BY REF,TEST6G_ULT BY REF)
! DUP KEY
IF ((ULT_CONTROL::ULT_FSI <> '*') AND (ULT_CONTROL::ULT_FSI <> 'N')) THEN
PRINT ULT_CONTROL::ULT_FSI
GOTO 999

END IF
IF (ULT_CONTROL::ULT_FSI='N') THEN
GOTO 999
END IF
! RDM INSERT FIRST TEST6G
ULT_CONTROL::ULT_OBJECT NAME='TEST6G'
ULT_CONTROL::ULT_OPERATION='LVIFU-'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6G BY REF,
ULT_DATE_STAMP BY REF,TEST6G_ULT BY REF)
! DUP KEY

```

```

IF ((ULT_CONTROL::ULT_FSI <> '*') AND (ULT_CONTROL::ULT_FSI <> 'N')) THEN
 PRINT ULT_CONTROL::ULT_FSI
 GOTO 999

END IF

IF (ULT_CONTROL::ULT_FSI='N') THEN
GOTO 999
END IF

! RDM INSERT NEXT TEST6G
ULT_CONTROL::ULT_OBJECT_NAME='TEST6G'
ULT_CONTROL::ULT_OPERATION='LVINU-'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6G BY REF,
ULT_DATE_STAMP BY REF,TEST6G_ULT BY REF)
! DUP KEY
IF ((ULT_CONTROL::ULT_FSI <> '*') AND (ULT_CONTROL::ULT_FSI <> 'N')) THEN
 PRINT ULT_CONTROL::ULT_FSI
 GOTO 999

END IF

IF (ULT_CONTROL::ULT_FSI='N') THEN
GOTO 999
END IF

! RDM INSERT LAST TEST6G
ULT_CONTROL::ULT_OBJECT_NAME='TEST6G'
ULT_CONTROL::ULT_OPERATION='LVILU-'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6G BY REF,
ULT_DATE_STAMP BY REF,TEST6G_ULT BY REF)
! DUP KEY
IF ((ULT_CONTROL::ULT_FSI <> '*') AND (ULT_CONTROL::ULT_FSI <> 'N')) THEN
 PRINT ULT_CONTROL::ULT_FSI
 GOTO 999

END IF

IF (ULT_CONTROL::ULT_FSI='N') THEN
GOTO 999
END IF

! RDM INSERT PRIOR TEST6G
ULT_CONTROL::ULT_OBJECT_NAME='TEST6G'
ULT_CONTROL::ULT_OPERATION='LVIPU-'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6G BY REF,
ULT_DATE_STAMP BY REF,TEST6G_ULT BY REF)
! DUP KEY

```

```

IF ((ULT_CONTROL::ULT_FSI <> '*') AND (ULT_CONTROL::ULT_FSI <> 'N')) THEN
 PRINT ULT_CONTROL::ULT_FSI
 GOTO 999

END IF

IF (ULT_CONTROL::ULT_FSI='N') THEN
 GOTO 999
END IF

800 DO UPDATE:
!800 DO_UPDATE: RDM UPDATE TEST6D
ULT_CONTROL::ULT_OBJECT_NAME='TEST6D'
ULT_CONTROL::ULT_OPERATION='LVU---'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6D BY REF,
 ULT_DATE_STAMP BY REF,TEST6D_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 CALL CSVBERROR(ULT_CONTROL)
END IF

 900 DO_DELETES:
!900 DO_DELETES: RDM DELETE TEST6E
ULT_CONTROL::ULT_OBJECT_NAME='TEST6E'
ULT_CONTROL::ULT_OPERATION='LVD---'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6E BY REF,
 ULT_DATE_STAMP BY REF,TEST6E_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 CALL CSVBERROR(ULT_CONTROL)
END IF

! RDM DELETE ALL TEST 6F
ULT_CONTROL::ULT_OBJECT_NAME='TEST6F'
ULT_CONTROL::ULT_OPERATION='LVD--*'
CALL CSVIPLVS(ULT_CONTROL BY REF,TEST6F BY REF,
 ULT_DATE_STAMP BY REF,TEST6F_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 CALL CSVBERROR(ULT_CONTROL)
END IF

 950 DO_MISCELLANEOUS
!950 DO_MISCELLANEOUS: RDM RELEASE
ULT_CONTROL::ULT_OPERATION='LVR---'
CALL CSVIPLVS(ULT_CONTROL BY REF,ULT_CONTROL BY REF,ULT_CONTROL BY REF,
 ULT_CONTROL BY REF)

```

```

IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 GOTO 999
END IF
! RDM MARK TEST1 AT TEST1_MARK
ULT_CONTROL::ULT_OBJECT_NAME='TEST1'
ULT_CONTROL::ULT_CONTEXT=TEST1_MARK
ULT_CONTROL::ULT_OPERATION='LVM---'
CALL CSVIPPLVS(ULT_CONTROL BY REF,TEST1 BY REF,
ULT_DATE_STAMP BY REF,TEST1_ULT BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
CALL CSVBERROR(ULT_CONTROL)
ELSE
TEST1_MARK_ULT_CONTROL::ULT_CONTEXT
END IF
! RDM COMMIT
ULT_CONTROL::ULT_OPERATION='LVC---'
CALL CSVIPPLVS(ULT_CONTROL BY REF,ULT_CONTROL BY REF,ULT_CONTROL BY REF,
ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 GOTO 999
END IF ! RDM RESET
ULT_CONTROL::ULT_OPERATION='LVA---'
CALL CSVIPPLVS(ULT_CONTROL BY REF,ULT_CONTROL BY REF, ULT_CONTROL BY REF,
ULT_CONTROL BY REF)
IF (ULT_CONTROL::ULT_FSI <> '*') THEN
 GOTO 999
END IF

999

END

```

# Index

## A

- access modes, permitted uses of
  - during SINON 336
- access-control parameter (PDML) 232
- adding primary records,
  - understanding how 215.
  - See also* ADD-M
- adding related records,
  - understanding how 219.
  - See also* ADDVA, ADDVB, and ADDVC
- ADD-M, PDML command
  - successful use of 361
  - using to add a record to a primary data set 238
- ADDVA, PDML command
  - efficiency of use 366
  - understanding how to use 219, 222
  - using to add a related record 242
- ADDVB, PDML command
  - efficiency of use 366
  - understanding how to use 219, 221
  - using to add a related record 249
- ADDVC, PDML command
  - efficiency of use 366
  - understanding how to use 219
  - using to add a related record 256
- ADDVR, PDML command 263
- ALL
  - using with DELETE
    - DBAID 53
  - using with DELETE (BASIC, FORTRAN, and COBOL) 169

- application program objects
  - modules, linking 348
- area-length parameter, PDML 232
- area-length, calculating size of 275
- ASI. *See* Attribute Status Indicator (ASI)
- assigning values to keys 32
- AT, using with
  - GET
    - BASIC 185
    - COBOL 180
    - DBAID 64
    - FORTRAN 185
    - MARK 132
- attribute list, displaying in DBAID 98
- Attribute Status Indicator (ASI)
  - definition of 117
  - example of generation
    - BASIC 119
    - COBOL 118
    - FORTRAN 118
  - values 119, 120

## B

- BASIC
  - data item descriptions 107
  - error handlers 144
  - listing of CSVBERROR 146
  - writing the program naming statement 108
- BASIC program
  - executing 149
  - linking 151
  - logic statements
    - controlling database recovery 140
    - handling error conditions 144
    - modifying rows 135, 136, 138
    - retrieving rows 123
    - signing off of RDM 122
    - signing on to RDM 122
  - running the RDM preprocessor for 149
  - sample RDM program 407, 409

## BASIC RDML statements

- COMMIT 167
  - DELETE 169
  - FORGET 173
  - INCLUDE 153
  - INCLUDE ULT-CONTROL 164
  - INSERT 190
  - MARK 195
  - RELEASE 198
  - RESET 200
  - SIGN-OFF 202
  - SIGN-ON 205
  - UPDATE 210
  - using hyphens and  
underscores in 104
- BEGN, for reading primary  
records 216

**C**C program, example PDM  
program 374

## CALL statements

- conventions for using 356
- list of parameters
- BASIC 355
- COBOL 354
- FORTTRAN 355

## closing views in DBAID 85

CNTRL, using PDML command  
269

## COBOL

- data item descriptions 106
- error conditions
- handling 142
- preventing loops in 143
- using NOT FOUND with 142
- writing the environment division  
109

COBOL procedure division  
statements

- controlling database recovery  
140
- handling error conditions 141
- modifying rows 135, 136, 138
- retrieving rows 123
- signing off of RDM 122
- signing on to RDM 122

## COBOL programs

- example PDM program 371
- executing 149
- linking 151
- running the RDM preprocessor  
for 149

## sample RDM program 375

COBOL RDM statements,  
INCLUDE 110

## COBOL RDML statements

- COMMIT 167
- DELETE 169
- FORGET 173
- GET 176
- INCLUDE 153
- INCLUDE ULT-CONTROL 163
- INSERT 190
- MARK 195
- RDML statement format 103
- RELEASE 198
- RESET 200
- SIGN-OFF 202
- SIGN-ON 205
- UPDATE 210

using dashes and underscores  
in 103

## code-directed reading 370

column descriptions, displaying  
57

## column descriptors, displaying 46

column number, specifying with  
BY-LEVEL  
DBAID 44column-name, specifying with  
DBAID

## COLUMN-DEFN 46

## COLUMN-TEXT 51

## FIELD-DEFN 57

## FIELD-TEXT 60

COMIT PDML command, using  
273command parameter, PDML 232  
commandsDBAID. *See* DBAID commandsPDML. *See* PDML commandsRDML. *See* RDML statements

## warning about coding RDML

and PDML commands in  
one program 347



- comments
  - displaying for column in a view
    - 51, 59
  - how they are shown by the
    - RDM preprocessor 110
- COMMIT, RDML statement
  - issuing in DBAID 93
  - using to control database
    - recovery 140
  - using to modify rows 134
- compiled database description
  - file
    - defining a logical name for
      - VMS 286, 317, 319, 323
- compiled database description
  - file, defining a logical name
    - for
      - VMS 286
- compiling RDM programs 149
- CONST key 32
- CSI\_RMS\_RU\_ON 140
- CSV\_SETUP\_REALM
  - description of 206
  - using to specify data set modes
    - 206
- CSVBERROR listing 146
- CSVDERERROR listing 145

## D

- data
  - positioning of after an error
    - COBOL 143
    - FORTTRAN and BASIC 148
  - updating in DBAID 95
  - validating 115
- Data Division
  - coding for COBOL 153
  - where to code it in your
    - program 153
- data item binding 233
- data item descriptions
  - BASIC 107
  - COBOL 106
  - FORTTRAN 107

- data item lists
  - data items you may not refer to
    - 360
  - determining efficiency of 366
  - parameter for 232
- data item, those you may not
  - refer to in a data item list
    - 360
- data recovery, controlling using
  - COMMIT and RESET 33
- data set access modes
  - effects of on other tasks 337
  - permitted uses of 336
- data set navigation
  - for primary records 215
  - for related records 223
- data sets
  - closing 214
  - closing using SINOF PDML
    - command 328
  - opening 214
  - opening using SINON PDML
    - command 332
  - specifying open modes for 206
- data-area parameter, PDML 232
- database
  - defining before executing your
    - RDM program 151
  - initializing 358
  - recovery, controlling 140
  - terminating 358
- data-item binding 369
- data-item-list parameter,
  - keywords 233
- data-set parameter, PDML 232
- date, size of the area length 275
- DBAID
  - accessing 36
  - accessing the SUPRA HELP
    - facility when using 41
  - automatic COMMIT facility,
    - disabling 45
  - automatic RESET facility,
    - disabling 77
  - signing on to 40
  - using to test views 34

## DBAID commands

- \* 42
- = 42
- BYE 43
- BY-LEVEL 43
- CAUTIOUS 45
- COLUMN-DEFN 46
- COLUMN-TEXT 51
- COMMIT 52
- DELETE 53
- ERASE 56
- FIELD-DEFN 56
- FIELD-TEXT 59
- FORGET 61
- GET
  - examples 66
  - using 62
  - using with nonunique keys 63
  - using with unique keys 63
- GO 68
- INSERT 72
  - inserting a view to an RMS data set 75
  - inserting nonuniquely keyed values 72
  - using MASS with 73, 75
- KEEP 77
- LINESIZE 78
- MARK 79
- MARKS 80
- OPEN 81
- PAGESIZE 83
- PRINT-STATS 84
- RELEASE 85
- RESET 86
- SHOW-NAVIGATION 87
- SIGN-OFF 88
- SIGN-ON 89
- STATS 90
- STATS-OFF 91
- STATS-ON 92
- SURE 93
- UNDEFINE 94
- UPDATE 95
- USER-LIST 98
- VIEW-DEFN 99
- VIEWS 101
- VIEWS-FOR-USER 102

## deadlock

- effect on performance 352
- preventing 352

deadly embrace. See deadlock

declaration statements, coding for BASIC and FORTRAN 153

default error-handlers 141

DEFINE command

- for compiled database description 286, 317, 319, 323

DEFINE command, for compiled database description 286

defining, program data 109

DELETE ALL, RDML statement, using to delete relationships 136

DELETE, RDML statement

- BASIC 169
- COBOL 169
- considerations for using 169
- FORTRAN 169
- using GET before performing 136
- using to delete relationships 137
- using to modify rows 136
- using with entities that are related 137

deleting

- entities 137
- primary records 218
- related records 226
- relationships 137

DEL-M, PDML command

- caution about using with RDNXT 277, 362
- understanding how to use 218
- using 276

DELVD, PDML command

- using 278
- using while processing related data sets with RDNXT 293

Destination Directory, specifying when running an RDM program 152

device failure recovery 23

Directory, functions of 23

- disabling
  - DBAID automatic COMMIT facility 45
  - DBAID automatic RESET facility 77
- displaying
  - attribute list for a user view 98
  - comments for column in a view 51, 59
  - condensed description of a view 99
  - current line-size setting 78
  - current statistics for views 90
  - field names 43
  - number of characters in a line 78
  - page-size setting 83
  - views currently active 101
- DUP KEY
  - using if keys you are inserting already have values 138
  - using in FORTRAN and BASIC error handlers 147
  - with INSERT 192

## E

- ELSE
  - with FORGET 175
  - with GET 180
  - with INSERT 192
- END ERROR HANDLER
  - with INCLUDE 158
  - with INCLUDE ULT-CONTROL 165
- END IF
  - with GET 188
  - with INSERT 190
- END., reference parameter 363
- Endp parameter, PDML 232
- enrolling program in the SUPRA Directory 108
- entities
  - adding 139
  - deleting 137
- environment division, writing for COBOL 109

- error handlers
  - default for BASIC 144
  - default for FORTRAN 144
  - using to handle unsuccessful RDM functions 134
  - when SUPRA uses a default error handler for FORTRAN and BASIC 144
  - writing for FORTRAN and BASIC 144
- error handling, writing for COBOL 141
- ERROR-ON, with COBOL error handler 141
- ERROR-ON-ULT-CONTROL 142
- errors
  - handling 134
  - handling in logical units of work 351
  - impacting positioning when using GET 134
- examples. *See* PDM program, examples; and RDM program, samples
- exiting DBAID 43

## F

- fatal embrace. *See* deadlock
- field names, displaying in DBAID 43
- files, SPECTRA central and personal 24
- FIRST, using with GET
  - COBOL 177
  - FORTRAN and BASIC 184
- INSERT 190
- INSERT to control row placement 138
- FOR UPDATE, using with GET
  - COBOL 178
  - FORTRAN and BASIC 185
- FORGET, RDML statement
  - BASIC 173
  - COBOL 173
  - FORTRAN 173

**FORTRAN**

- data item descriptions 107
  - error handlers 144
  - error handlers, using DUP KEY and NOT FOUND statements 147
  - listing of CSVDError 145
  - writing Declaration Statements 153
  - writing the program naming statement 108
- FORTRAN program logic**
- controlling database recovery 140
  - handling error conditions 144
  - modifying rows 135, 136, 138
  - retrieving rows 123
  - signing off of RDM 122
  - signing on to RDM 122
- FORTRAN programs**
- example PDM program 372
  - linking 151
  - running the RDM preprocessor for 149
  - sample RDM program 391, 395
- FORTRAN RDML statements**
- COMMIT 167
  - DELETE 169
  - FORGET 173
  - INCLUDE 110
  - INCLUDE 153
  - INCLUDE ULT-CONTROL 164
  - INSERT 190
  - MARK 195
  - RELEASE 198
  - RESET 200
  - SIGN-OFF 202
  - SIGN-ON 205
  - UPDATE 210
  - using hyphens and underscores in 104
- FSI. See Function Status Indicator (FSI)**
- Function Status Indicator (FSI)**
- 116

**G**

- gathering statistics in DBAID 92
- generic reads
  - considerations for using 129
  - performing 127
- GET FIRST
  - examples 123
  - using 123, 124
- GET LAST, using 124
- GET NEXT
  - examples 123
  - using 123
- GET, RDML statement
  - BASIC 183
  - COBOL 176
  - FORTRAN 183
  - using to control record holding 133

**H**

- held status, what to do if you receive 352

**I**

- identification division, writing for
  - COBOL 108
- INCLUDE ULT-CONTROL, RDML statement
  - BASIC 164
  - COBOL 163
  - FORTRAN 164
  - where to include in your COBOL program 111
- INCLUDE, RDML statements
  - BASIC 110, 153
  - COBOL 110, 153
  - FORTRAN 110, 153
- index masking 317
- index Masking 317
- initializing
  - PDM programs 351
  - SUPRA database 358

INSERT, RDML statement  
 BASIC 190  
 COBOL 190  
 coding an UPDATE statement  
   after using 138  
 FORTRAN 190  
 using to modify rows 138  
 using with null values 139  
 issuing  
   COMMIT in DBAID 93  
   RDM DELETE in DBAID 53, 54  
   RDM INSERT in DBAID 72  
   RDM RELEASE in DBAID 85  
   RDM RESET in DBAID 86  
   sweeping GETS in DBAID 68

## K

keys  
   assigning values to 32  
   changing to enable use of the  
     UPDATE statement on the  
       row 135  
   nonunique 32  
   retrieving rows with 123  
   retrieving rows without 123  
   secondary 127  
   unique 31  
   using wildcard characters in  
     127  
 keywords, in data-item-list  
   parameter 233

## L

LAST, using with  
   INSERT to control row  
     placement 138  
 LINE SIZE setting, displaying in  
   DBAID 78  
 linking your application program  
   348  
 linking your RDM program 151  
 linkpath parameter, PDML 232  
 listing all open MARKs in DBAID  
   80  
 LKxx reference parameter 363  
 LOCK, to hold records 272

logical names  
   compiled database description  
     file (VMS) 286, 317, 319,  
     323  
 logical statistics, printing in  
   DBAID 84  
 logical unit of work  
   definition of 350  
   handling errors in 353  
   implementing 351  
   preventing deadlocks between  
     352  
   reserving resources for 350  
 loops, preventing in COBOL error  
   handlers 143

## M

MANTIS 24  
 MARK statement  
   BASIC 195  
   COBOL 195  
   considerations for using 195  
   FORTRAN 195  
   using to save position of a row  
     for later access 132  
 marking the current position of a  
   view in DBAID 79  
 MARKL, using 283  
 MARKS  
   listing 80  
   removing in DBAID 61  
 MASK option for index searches  
   317  
 metadata, definition 23  
 mode. *See* data set access  
   modes

## N

NEXT, using with  
   INSERT to control row  
     placement 138  
 node name parameter 232  
 NOT FOUND  
   including in your program 125  
   using with an unsuccessful  
     RDM function 134  
   using with COBOL error-  
     handlers 142  
 null values, inserting 139

**O**

ON ERROR, in FORTRAN and  
    BASIC error handlers 144  
OPCOM, PDML command 285  
opening views in DBAID 81  
option parameter, PDML 232

**P**

page-size setting, displaying in  
    DBAID 83  
parameters, PDML. *See* PDML  
    parameters  
PDM (Physical Data Manager)  
    function of 23  
    supported recovery techniques  
        23  
PDM program  
    checking the status parameter  
        362  
    examples  
        C 374  
        COBOL 371  
        FORTRAN 372  
        PL/I 373  
    improving program efficiency  
        366  
    initialization and termination  
        requirements 358  
    initialization of 351  
    linking 348  
    managing 354  
    requirements for terminating  
        358  
    testing 368  
PDML (Physical Data  
    Manipulation Language)  
    command categories 229  
    command parameters 232  
    commands, brief description  
        229  
    definition of 213  
    general format 227

    understanding how to use  
        adding a primary record 215  
        adding a related record 219  
        deleting a primary record 218  
        deleting a related record 226  
        opening/closing data sets 214  
        reading a primary record 215  
        reading a related record 215  
        updating a primary record  
            218  
        updating a related record 225  
PDML commands  
    ADD-M 238  
    ADDVA 242  
    ADDVB 249  
    ADDVC 256  
    ADDVR 263  
    CNTRL 269  
    COMIT 273  
    DEL-M 276  
    DELVD 278  
    MARKL 283  
    OPCOM 285  
    RDNXT 289  
    READD 294  
    READM 300  
    READR 303  
    READV 309  
    READX 314  
    RESET 324  
    RQLOC 326  
    SINOF 328  
    SINON 332  
    WRITM 338  
    WRITV 342  
PDML parameters, by PDML  
    command 232  
performance, effect of record  
    holding on 132  
Physical Data Manager. *See*  
    PDM and PDM program  
Physical Data Manipulation  
    Language. *See* PDML  
physical key  
    adding with ADD-M 239, 361  
    changing 338  
    deleting with DEL-M 276  
physical statistics, printing in  
    DBAID 84

- PL/I program, example PDM program 373
- position of a view, DBAID 79
- ppppLKxx 360
- preprocessor. *See* RDM preprocessor
- primary data set processing 360
- primary record
  - adding in PDM programs 215
  - adding using ADD-M 238
  - deleting in PDM programs 218
  - deleting using DEL-M 276
  - reading in PDM programs 215
  - updating in PDM programs 218
- printing current statistics in DBAID 91
- printing logical statistics in DBAID 84
- printing physical statistics in DBAID 84
- PRIOR, using with
  - INSERT to control row placement 138
- procedure division, writing for COBOL and FORTRAN 122
- program data, defining 109
- program logic statements, writing for BASIC and FORTRAN 167
- program naming statement
  - FORTRAN 108
- program naming statement, writing for BASIC and FORTRAN 108
- program, enrolling in the SUPRA Directory 108. *See also* PDM program or RDM program

## Q

- qualifier parameter, PDML 232

## R

- RDM (Relational Data Manager)
  - how it uses your program to manipulate data 26
  - preprocessor, functions of 22
  - retrieval and maintenance operations 22
  - understanding how it works 26
  - using RDML to access data 26
- RDM access paths, verifying the accuracy of using DBAID 87
- RDM preprocessor
  - function of 26
  - how it comments out statements 110
  - running 149
- RDM program
  - compiling 149
  - how RDM checks for currency 121
  - implementing and executing 149
  - linking 151
  - samples
    - BASIC 407, 409
    - COBOL 375, 378
    - FORTRAN 391, 395
  - testing using DBAID 35
- RDM statements
  - SIGN-OFF 30
  - SIGN-ON 30
- RDML (Relational Data Manipulation Language)
  - understanding 30
  - understanding how to retrieve rows using keys 31
  - understanding required columns 32
  - understanding sequential retrieval using GET 31
  - unique and nonunique key values 31
  - using compound nonuniques
    - keys to retrieve data 32
  - using nonunique keys to retrieve data 32
  - using unique keys to retrieve data 31

RDML (Relational Data  
Manipulation Lanuguage)  
special function statements  
(MARK, FORGET,  
RELEASE) 33  
RDML statement format  
BASIC 104  
COBOL 103  
FORTRAN 104  
RDML statements  
COMMIT  
BASIC 167  
COBOL 167  
FORTRAN 167  
DELETE  
BASIC 169  
COBOL 169  
FORTRAN 169  
FORGET  
BASIC 173  
COBOL 173  
FORTRAN 173  
GET  
BASIC 183  
COBOL 176  
FORTRAN 183  
INCLUDE  
BASIC 153  
COBOL 153  
FORTRAN 153  
INCLUDE ULT-CONTROL  
BASIC 164  
COBOL 163  
FORTRAN 164  
INSERT  
BASIC 190  
COBOL 190  
FORTRAN 190  
MARK  
BASIC 195  
COBOL 195  
FORTRAN 195  
RELEASE  
BASIC 198  
COBOL 198  
FORTRAN 198  
RESET  
BASIC 200  
COBOL 200  
FORTRAN 200  
SIGN-OFF  
BASIC 202

COBOL 202  
FORTRAN) 202  
SIGN-ON  
BASIC 205  
COBOL 205  
FORTRAN 205  
UPDATE  
BASIC 210  
COBOL 210  
FORTRAN 210  
RDNXT  
caution about using with DEL-M  
277  
using 289  
using to read a primary record  
215  
using to read a related file 224  
when to use 367  
READ, to hold records 272  
READD  
issuing after a DELVD 282  
using 294  
reading primary records 215  
reading related records 223  
READM  
understanding how to use for  
primary records 215  
using 300  
using when deleting primary  
records 218  
using when updating primary  
records 218  
READR  
issuing after a DELVD 282  
using 303  
READV  
issuing after a DELVD 282  
using 309  
READX  
using 314  
using to read primary records  
216  
using to read related records  
224  
REALM  
with SINOF 329  
with SINON 333  
REBD  
using when reading a primary  
record 216  
recompiling, when necessary 121



- record code, changing using
  - ADDVR 263
- record holding
  - effect on performance 132
  - how to use GET statement to control it 133
- recovery
  - controlling 140
  - from device 23
  - shadow recording 23
  - system logging 23
  - task level 23
- reference parameter
  - END. 363
  - LKxx 363
  - RRN 363
- reference parameter values
  - before and after issuing a command 364
  - significant changes in after issuing commands 365
- reference parameter, PDML 232
- reissuing DBAID commands 42
- related data set
  - processing 362
  - using code directed reading with 370
- related record
  - adding in PDM programs 219
  - adding using ADDVA 219, 222
  - adding using ADDVB 219, 221
  - adding using ADDVC 219
  - consideration for successful processing 362
  - deleting in PDM program 226
  - deleting using DELVD 278
  - reading in PDM programs 223
  - updating in PDM programs 225
- related record number. *See* RRN
- Relational Data Manager. *See* RDM
- relationships
  - adding with entities in one operation 139
  - deleting 137
- RELEASE statement
  - BASIC 198
  - COBOL 198
  - considerations for using 198
  - FORTRAN 198
- removing MARKS in DBAID 61
- RESET
  - PDML command
    - issuing after a COMMIT 275
    - using to handle errors in a logical unit of work 353
  - RDML statement
    - using to control database recovery 140
- RESET statement
  - BASIC 200
  - COBOL 200
  - considerations for using 200
  - FORTRAN 200
- RESET, PDML command
  - using 324
- retrieving rows. *See* rows, retrieving
- RMS files, recovering 140
- RMS Recovery Unit Journaling 140
- rolling back database updates 86
- rows
  - controlling the placement of
    - when using INSERT 138
  - inserting, in DBAID 72
  - modifying 134
    - using COMMIT when modifying 134
    - using DELETE 136
    - using INSERT 138
    - using UPDATE 135
  - retrieving
    - in DBAID using GET 62
    - in DBAID using GO 68
    - using COMMIT when modifying 134
    - using partial keys 127
    - using wildcard characters 127
    - with keys 123
    - without a key 123
- RQLOC PDML command 326
- RRN
  - reference parameter 363
  - repositioning of after using DELVD 278
  - sources when supplied to READD 299

**S**

- serial retrieval
  - caution about when adding and deleting 362
  - performing using RDNXT 289
- Serial retrieval
  - description of 215
- shadow recording 23
- shared holds 272
- signing off of DBAID 88
- signing on to DBAID 89
- SIGN-OFF RDML statement
  - DBAID 88
- SIGN-OFF statement
  - BASIC 202
  - COBOL 202
  - considerations for using 202
  - FORTRAN 202
- SIGN-ON RDML statement
  - DBAID 89
- SIGN-ON statement
  - BASIC 205
  - COBOL 205
  - considerations for using 205
  - FORTRAN 205
- SINOF PDML command 328
- SINON access modes
  - effects of on tasks 337
  - permitted uses 336
- SINON PDML command 332
- specifying the number of lines on a screen/page in DBAID) 83
- SPECTRA 24
- statements
  - PDML. *See* PDML commands
  - RDML. *See* RDML statements
- statistics
  - displaying for views in DBAID 90
  - gathering in DBAID 92
  - printing in DBAID 91
- status indicators for RDML
  - commands 116
- status parameter, PDML,
  - checking 359
- statuses, PDM
  - received at SINOF 331
  - received at SINON 335

## SUPRA Server

- communicating with using
  - CALL statements 354
- default error handler 142
- Directory 23
- exit handler 358
- HELP facility 41
- tools for programmers
  - UNIX 21
  - VMS 20, 23
- SUPRA Server databases
  - initialization and termination
    - requirements for 358
  - SUPRAD 26
- sweeping GETS, issuing in
  - DBAID 68
- system log file, writing records to
  - using MARKL 283
- system logging 23

**T**

- task access modes, compared
  - with data set access modes 336, 337
- task level recovery 23
- task log file 324
- task management 358
- testing
  - PDML programs 368
  - RDML programs using DBAID 35
- testing views 34
- TLR. *See* task level recovery

**U**

- ULT-CONTROL, coding with an
  - INCLUDE statement
- BASIC 112
- COBOL 111
- FORTRAN 112
- unsuccessful RDM functions,
  - handling 134

UPDATE RDML statement  
   considerations for using on a  
     view key 135  
   using after an INSERT  
     statement to update  
       columns 139  
   using to modify rows 135  
   using with null values 139  
 UPDATE statement  
   BASIC 210  
   COBOL 210  
   considerations for using 210  
   FORTRAN 210  
 updating  
   data values in DBAID 95  
   primary records 218  
   related records 225  
 user view 28  
 USING clause 123

## V

validating data 115  
 Validity Status Indicators 120  
 VAX compiler, running to execute  
   an RDM program 150  
 verifying RDM access paths in  
   DBAID 87  
 view  
   creating a view for your  
     program 110  
   definition 27  
   ULT-CONTROL 111, 112, 113  
   using an established view for  
     your program 110  
   using subsets of 28  
 view descriptors 100  
 view keys  
   how to code your INSERT  
     when a view is uniquely  
       keyed 138  
   using the UPDATE statement  
     on 135  
 view-name  
   with BY-LEVEL (DBAID) 43  
   with COLUMN-DEFN (DBAID)  
     46  
   with COLUMN-TEXT (DBAID)  
     51  
   with DELETE  
     BASIC 169  
     COBOL 169  
     FORTRAN 169  
   with DELETE (DBAID) 53  
   with FIELD-DEFN (DBAID) 56  
   with FIELD-TEXT (DBAID) 59  
   with GET  
     BASIC 185  
     COBOL 178  
     DBAID 64  
     FORTRAN 185  
   with GO (DBAID) 68  
   with INCLUDE  
     BASIC 158  
     COBOL 154, 158  
     FORTRAN 158  
   with INSERT  
     BASIC 191  
     COBOL 191  
     DBAID 73  
     FORTRAN 191  
   with MARK  
     BASIC 195  
     COBOL 195  
     DBAID 79  
     FORTRAN 195  
   with OPEN  
     DBAID 81  
   with RELEASE  
     BASIC 198  
     COBOL 198  
     DBAID 85  
     FORTRAN 198  
   with SHOW-NAVIGATION  
     DBAID 87  
   with STATS  
     DBAID 90  
   with UNDEFINE  
     DBAID 94  
   with UPDATE  
     BASIC 210  
     COBOL 210  
     DBAID 95  
     FORTRAN 210  
   with VIEW-DEFN  
     DBAID 99  
 views  
   displaying 101

virtual view, removing the name  
and definition of 94  
VMS RMS Recovery Unit  
Journalling 140  
VSI. *See* Validity Status  
Indicators

## W

wildcard character  
retrieving rows with 127  
using \* 127  
using = 127  
writing program naming  
statement  
BASIC 108  
FORTRAN 108

writing the environment division  
COBOL 109  
WRITM  
using 338  
using while processing primary  
data sets with RDNXT 293  
what you should do before  
using 218  
WRITV  
using 342  
using processing related data  
sets with RDNXT 293